
**Informatique
pour Ingénieurs de gestion**

Bloc 2

Année académique
2021-2022

Michaël Schyns

Table des matières

I	Internet	7
1	Les langages HTML et XHTML (rappels et suite)	7
1.1	Principe de fonctionnement	7
1.2	Autres langages	8
1.3	Balises HTML	9
1.4	Structure de base d'un document HTML/XHTML	9
1.4.1	Commentaires	10
1.5	Retour à la ligne	10
1.6	Formatage du texte	11
1.7	Caractères spéciaux	13
1.8	Liens	13
1.8.1	Ancres	14
1.9	Images	15
1.10	Vidéo	16
1.11	Bande son	17
1.12	Page dans une page	17
1.13	Tabulations et listes	18
1.14	Les tables	19
2	Les feuilles de style CSS	21
2.1	Insertion de règles CSS	21
2.1.1	Feuille de style externe	21
2.1.2	Feuille de style interne	22
2.1.3	Inline	23
2.2	Quelques déclarations utiles	23
2.2.1	Formater du texte	23
2.2.2	Marges, bordures et padding	24
2.2.3	Arrière plan	25
2.2.4	Tables	26
2.2.5	Liens	26
2.2.6	Listes	27
3	Structurer une page avec le CSS	28
3.1	Éléments flottants	30
3.2	Overflow	31
3.3	Exemple de structure de page	32
4	Les formulaires HTML	35
4.1	Objets d'un formulaire	36
II	Systèmes d'information dynamiques	42

5	Les bases du langage PHP	42
5.1	Principes et fonctionnement	42
5.2	Structure de base	43
5.2.1	Commentaires	43
5.3	Fonction ECHO	43
5.4	Les variables	44
5.4.1	Le langage binaire	44
5.4.2	Définition de variables	45
5.5	Chaînes de caractères	45
5.6	Opérateurs de base	46
5.7	Instructions de test IF	47
5.7.1	Opérateurs de comparaison	49
5.7.2	Opérateurs logiques	50
5.8	Les boucles	50
5.8.1	Boucle while	50
5.8.2	Boucle for	51
5.9	Les tableaux - vecteurs	53
5.10	Fonctions utiles en PHP	54
5.10.1	Date et temps	54
5.10.2	Precision	54
5.10.3	Chaînes de caractères	54
5.10.4	Aléatoire	54
5.10.5	Tableaux	55
5.11	Récupération des données d'un formulaire	55
5.12	Données persistantes : cookies et sessions	56
5.12.1	Ecriture d'un cookie	57
5.12.2	Lecture d'un cookie	57
5.12.3	Création de sessions	58
5.13	E-commerce	60
5.13.1	Création de compte	60
5.13.2	Création d'un bouton de paiement	61
5.13.3	Acheter	62
5.13.4	Gestion des comptes	64
6	Bases de données et PHP	65
6.1	MySQL et PHPMyadmin	65
6.1.1	Création de tables	66
6.1.2	Insertion de données dans une table	67
6.1.3	Consultation de table	67
6.2	SQL	68
6.2.1	Instructions SQL	68
6.3	SGBD et PHP	70
6.4	Fonctions MySQL	70

III Java 75

7	Introduction	75
7.1	Les langages de programmation et le Java	75
7.2	L'environnement de programmation et Eclipse	76
7.3	Installation du JDK et d'Eclipse	76
7.3.1	Installation du JDK	76
7.3.2	Installation d'Eclipse	77
8	Les bases du langage Java	78
8.1	Démarrer avec Eclipse	78
8.1.1	Environnement Eclipse	80
8.2	Structure de base	82
8.2.1	Les commentaires	84
8.3	Les variables	84
8.3.1	Variables de type numérique	84
8.3.2	Variable de type caractère	85
8.3.3	Variable de type booléen	85
8.4	Chaînes de caractères	85
8.4.1	Conventions de nommage en Java	86
8.5	Afficher à l'écran	86
8.6	Opérateurs de base	88
8.6.1	Concaténation	88
8.6.2	Cast de variables	89
8.7	Récupération des données saisies par l'utilisateur	91
8.8	Instructions de test IF	93
8.8.1	Opérateurs de comparaison	94
8.8.2	Opérateurs logiques	94
8.9	Les boucles	95
8.9.1	Boucle while	95
8.9.2	Boucle for	96
8.10	Les tableaux à une dimension	96
8.11	Les tableaux multidimensionnels	98
8.12	Méthodes utiles en Java	99
8.12.1	Mathématiques	99
8.12.2	Chaînes de caractères	99
8.12.3	Tableaux	100
8.12.4	Aléatoire	100
9	Lecture et écriture dans des fichiers	101
9.1	Exceptions	101
9.2	Récupérer les données contenues dans un fichier	103
9.3	Ecrire des variables dans un fichier	106

Introduction à l'informatique

2ème partie

Année 2021 - 2022

Préambule

Etre un gestionnaire c'est avant tout prendre des décisions. Or, prendre une bonne décision ne se fait pas sans information. Aujourd'hui, aucune entreprise ne peut survivre sans une bonne gestion de l'information : les *Technologies de l'Information et de la Communication* (TIC) sont les clefs du succès. En particulier, Internet et le Web sont devenus incontournables : Amazon, Google, Facebook... sont des exemples évidents d'entreprises surfant sur la vague des TIC. Néanmoins, cela est vrai également dans toutes les autres entreprises, de la plus petite à la plus grande, locale ou internationale : un gestionnaire, quel que soit son domaine, doit aujourd'hui être capable de collecter, structurer, analyser, traiter des masses de données pour pouvoir répondre aux nombreuses questions managériales auxquelles il est confronté.

Par ailleurs, face à la concurrence, cela ne suffit plus de simplement gérer les processus traditionnels d'une entreprise : il est aussi question d'innovation. Grâce aux technologies, de nouveaux produits et de nouveaux services sont envisageables. En outre, de nouveaux modèles économiques apparaissent ou améliorent les modèles traditionnels.

Un Ingénieur de Gestion, de par son profil, se distingue par sa capacité à utiliser des outils et technologies pointus basés sur des méthodes plus quantitatives et scientifiques. Ces compétences sont non seulement des atouts non négligeables dans la carrière professionnelle d'un Ingénieur de gestion, mais elles constituent également des prérequis pour d'autres cours de son curriculum. Ce cours, en complément de nombreux autres cours orientés "*Digital*" dans le cursus de l'Ingénieur de Gestion, se veut une préparation aux nouveaux défis et aux nouvelles opportunités auxquels le gestionnaire va désormais faire face. Nous sommes dans un nouveau monde où tout est possible ! Nous sommes à l'ère du "*Digital Business*". *Bienvenue à HEC-Liège, à Digital Business School.*

Ces notes, synthétiques, sont à destination des étudiants de bachelier en Ingénieur de Gestion (bloc 2). Elles concernent la seconde partie de la formation

en informatique et systèmes d'information de gestion.

Les matières présentées dans la première partie de ces notes sont des compléments aux matières vues en bloc 1. Les étudiants sont supposés maîtriser la matière du bloc 1. Néanmoins, dans le souci de fournir un support complet et homogène à tous les étudiants, la première partie reprend l'essentiel du cours de bloc 1 qui est jugé nécessaire. Les étudiants qui souhaitent plus d'informations sont invités à lire ou relire ce cours.

Ces notes sont un support et ne remplacent pas les notes prises au cours et aux répétitions. Le lecteur remarquera que ces notes de cours contiennent de nombreux exemples. Dans ces exemples, seul le code est présenté, et non le résultat de son interprétation. L'informatique ne peut pas s'apprendre uniquement par la lecture. Il est essentiel de tester par soi-même les exemples de code fournis et de s'assurer de leur compréhension totale. Des documents supplémentaires, dont les résolutions des exercices réalisés durant les cours, seront disponibles en ligne sur la plateforme **eCampus** pour compléter ces notes.

Ce cours permet déjà d'atteindre un bon niveau d'autonomie en informatique. Pour le lecteur curieux et désireux d'aller plus loin, le site d'apprentissage www.w3schools.org est une excellente référence pour en savoir plus sur les langages abordés dans ce cours (HTML, CSS, PHP) ainsi que d'autres. Pour le langage Java, le site <http://docs.oracle.com/javase/tutorial/java/> (en anglais) est également très utile.

Première partie

Internet

1 Les langages HTML et XHTML (rappels et suite)

1.1 Principe de fonctionnement

Nous ne reprendrons pas ici tout ce qui a été présenté l'an passé et le lecteur est invité à relire les notes de bloc 1. Néanmoins, vu l'importance du langage HTML pour la suite et certaines contraintes supplémentaires (XHTML) qu'il est souhaitable d'ajouter pour assurer une plus grande portabilité des pages, quelques rappels seront initialement réalisés.

Pour permettre le transfert et la visualisation correcte de fichiers entre des ordinateurs distants et différents, il était nécessaire de définir un nouveau type de document que tous ces ordinateurs seraient capables de lire et d'afficher. Ainsi est né le langage HTML (*Hyper Text Markup Language* ou encore littéralement en français *Langage de Balisage d'Hypertexte*). Toutes les pages du Web reçues par un navigateur sont en réalité des fichiers texte intégrant des commandes dans ce langage (ou un de ses dérivés). Il est d'ailleurs possible de voir, sur une page Web quelconque, le code correspondant (code source).

Notons que le HTML est un langage qui évolue. La version 4 a été longtemps le standard ces dernières années et est toujours très présente. Le HTML 5 a été approuvée par le W3C Consortium en octobre 2014 (pour plus d'informations, consultez le site www.w3.org). Ce langage est relativement souple. Par exemple, les marques peuvent aussi bien être écrites en minuscules qu'en majuscules ou même les deux. Cela ne pose aucun problème pour nos ordinateurs qui ont la puissance nécessaire pour considérer tous ces cas. Par contre, à l'ère de la mobilité, il n'en est pas de même pour les micro-ordinateurs (embarqués dans les voitures, les GSM, les PDA...). Dès lors, une version plus rigoureuse et plus facile à manipuler a été définie. L'objectif de cette version n'est pas d'enrichir les marques et les capacités du langage, mais bien uniquement de le rendre plus rigoureux. C'est pourquoi il a été décidé de le nommer XHTML, pour marquer la différence avec les précédentes versions du HTML. Le nom XHTML est une contraction de XML et de HTML. Le "X" signifie *eXtensible*. XML est un langage "rigoureux" de modélisation de données basé, comme pour le HTML, sur des marques/balises. Le langage XML est très important pour les entreprises car il permet de structurer des données en vue de les communiquer à des partenaires via Internet (et d'autres réseaux). Les améliorations qu'il apporte ne le rendent pas incompatible avec le HTML 5 et les navigateurs classiques peuvent donc toujours les comprendre. Principalement, on remarque que l'utilisation des minuscules est obligatoire en XHTML et que chaque marque doit être fermée. Il

y a d'autres changements importants, comme la dissociation de la présentation et du contenu par des fichiers de style. Dans ce cours, on tentera de prendre de bonnes habitudes en respectant le plus possible le XHTML.

1.2 Autres langages

Le HTML et le XHTML sont les langages de base compris par un navigateur. Il existe néanmoins d'autres langages utilisés par les développeurs de sites web.

Du côté du serveur, les pages Web peuvent être écrites en HTML à l'aide d'un logiciel d'édition (Notepad par exemple) et placées dans un répertoire du serveur en attendant qu'un internaute les demande. Chaque Internaute qui demande une telle page obtient le même code HTML et donc le même contenu s'affiche dans son navigateur. On parle de *pages statiques*.

Une autre approche est d'écrire des *pages dynamiques*. La page n'existe pas a priori sur le disque dur du serveur, mais on y trouve à la place un programme qui, lorsqu'il est exécuté, crée la page au vol pour l'envoyer à l'Internaute. La page étant créée au vol, son contenu peut être adapté instantanément en fonction de l'Internaute. Par exemple, ce sera le cas lors de l'utilisation d'un moteur de recherche comme Google. La page HTML reçue en réponse n'existait bien entendu pas à l'avance, mais a été créée au moment même pour tenir compte des mots clefs fournis par l'Internaute. Il existe différents langages pour écrire de tels programmes. Parmi les plus courants, on trouve l'ASP.NET de Microsoft et son concurrent direct le PhP (PhP : *Hypertext Processor*) de la communauté "Open". On rencontre aussi des programmes en Java.

Il est important de noter que le navigateur ne reçoit que le code HTML. Il ne voit jamais le code du programme qui l'a généré. Dans le cadre de ce cours, le langage PhP sera présenté comme langage principal pour le web. Une introduction au Java est également proposée comme langage pour application standalone, mais aussi web.

Côté navigateur, il est possible d'incorporer dans le code HTML des instructions à exécuter, cette fois non pas par le serveur, mais par l'ordinateur de l'Internaute. Cela permet de décharger le serveur de certaines opérations et d'accélérer/améliorer la lecture de certaines pages. Pour ce faire, le langage le plus connu est le "JavaScript". Par exemple, avant d'envoyer un formulaire sur le Web, le code javascript peut s'assurer que vous avez bien rempli les champs obligatoires. Son code est intégré dans le HTML et est donc visible par l'Internaute. À côté du Javascript, il est aussi possible d'intégrer de petits programmes "compilés" (on reçoit le programme directement exécutable, mais on ne voit pas le code) ou précompilé (dans une langage intermédiaire). L'exemple le plus connu est celui des programmes (applets) en langage Java.

Lié aux versions modernes du HTML, on parle aussi du "langage" CSS (*cascading style sheet*). Ce langage permet de définir des feuilles de style et donc la mise en page et certains éléments d'interactivité d'un site. Les possibilités de personnalisation de sites offertes par le langage CSS sont extrêmement vastes.

Le principe de ce langage et quelques propriétés de style intéressantes seront détaillées dans la section 2.

1.3 Balises HTML

Le HTML est, comme son nom l'indique, un langage de balisage, c'est à dire un langage spécialisé dans l'amélioration d'informations textuelles qui utilise des *balises* (ou *marques*). Des balises sont des éléments de code encadrant une portion de texte dont on veut modifier l'aspect. En général, les balises vont par paires : une balise de début dont la syntaxe est du type `<nom_de_la_marque>` ainsi qu'une balise de fin dont la syntaxe est du type `</nom_de_la_marque>`. Par exemple, il suffit d'encadrer un mot par les balises `` et `` pour que celui-ci apparaisse en gras sur la page Web créée.

Certaines balises vivent seules, c'est à dire qu'elles n'ont pas de balise de fin correspondante. C'est par exemple le cas de la balise permettant de passer à la ligne `
`. Dès lors, pour respecter la syntaxe du XHTML, on ajoute un `/` juste avant le `>` pour indiquer qu'il n'y aura pas de marque de fermeture. En HTML, le `/` n'est pas nécessaire.

Une balise peut avoir des *paramètres* (ou *attributs*) servant à spécifier sa signification. Ces arguments sont précisés dans la balise de début en utilisant la syntaxe suivante `<balise nom_de_la_marque paramètre="value">`. Les paramètres consistent donc en un nom et une valeur. Si plusieurs paramètres sont spécifiés, leur ordre n'a pas d'importance. Pour obtenir la liste complète des paramètres possibles pour une balise spécifique, nous conseillons au lecteur de consulter le site incontournable www.w3schools.org.

Notons que le navigateur a deux missions : récupérer les pages sur base d'une URL et les interpréter pour en donner une représentation graphique. Il est dès lors courant qu'une même balise ne donne pas exactement le même résultat à l'écran suivant le navigateur utilisé (cfr. www.w3.org pour des explications complètes).

1.4 Structure de base d'un document HTML/XHTML

Un document HTML correctement écrit commence par `<!DOCTYPE html>` et se termine par `</html>`. Ces instructions ont pour but d'indiquer au navigateur le type de document auquel s'attendre (i.e. HTML).

Le reste du document est composé de deux parties :

1. **L'en-tête** (entourée des balises `<head>...</head>`) : elle contient une série d'informations, des *méta-données*, sur le document qui seront utilisées par le navigateur et les serveurs sur Internet (les moteurs de recherche en particulier). Au minimum, on y trouvera le titre du document qu'on précise entre les balises `<title>... </title>`. Celui-ci sera affiché dans la barre du navigateur, dans l'historique de navigation, dans

les favoris ou dans des résultats de recherche, mais pas sur la page elle-même. Le type d'encodage sera également précisé dans l'en-tête à l'aide de la balise `<meta charset="UTF-8">`. A l'aide de la balise `meta`, il est également possible de définir des mots clés et une description pour la page Web créée, de spécifier l'auteur, de rafraîchir automatiquement la page toutes les trente secondes (voir http://www.w3schools.com/tags/tag_meta.asp pour plus d'informations). C'est également dans l'en-tête qu'on retrouve principalement les règles de style CSS applicables à la page.

2. **Le corps** (entouré des balises `<body>...</body>`) : celui-ci contient le texte qui sera affiché dans le navigateur (le contenu visible de la page).

Exemple 1 *Structure minimale d'un document HTML.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <title> Mon titre </title>
6 </head>
7 <body>
8   Le texte
9 </body>
10</html>
```

1.4.1 Commentaires

Notons qu'il est possible d'ajouter des commentaires à votre code HTML. Les commentaires sont du texte rédigé à l'intention des lecteurs du code, qui ne s'affiche pas sur la page résultat. Les commentaires sont particulièrement utiles lorsque le code est long et complexe car cela permet d'ajouter des explications. Commenter son code est une habitude fortement recommandée, surtout lorsque d'autres personnes que vous doivent le lire. Les commentaires doivent être entourés des balises `<!--` et `-->`.

1.5 Retour à la ligne

Le navigateur Internet ne tient pas compte des espaces, tabulations et retours à la ligne présents dans les documents HTML. Pour provoquer un retour à la ligne, on utilise la balise `
` (*break*) à l'endroit voulu.

Exemple 2 *Retour à la ligne*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title> Retour à la ligne </title>
5 </head>
6 <body>
7   Ma première ligne
8   Ma deuxième ligne? Eh non!
```

```
9 | <br/>
10 | Une ligne <br/>
11 | Une autre ligne <br/><br/>
12 | </body>
13 | </html>
```

Une autre possibilité consiste à séparer le texte en sections ou en paragraphes. Chaque section ou paragraphe commence à une nouvelle ligne. Pour définir une section, on utilise les balises `<section> ... </section>`. Le titre de la section peut être spécifié entre les balises `<header>...</header>`. Un bas de section contenant par exemple l'auteur, la date, un éventuel copyright peut être précisé à l'aide des balises `<footer>...</footer>`.

Exemple 3 *Section*

```
1 | <section>
2 |   <header>
3 |     <h1> WWF</h1>
4 |   </header>
5 |   WWF's mission is to stop the degradation of our planet's
   |   natural environment.
6 |   <footer>
7 |     Contact information: someone@example.com
8 |   </footer>
9 | </section>
```

Pour constituer un paragraphe, on utilise les balises `<p>...</p>`. Le paramètre `title="..."` permet de donner un titre au paragraphe qui s'affichera lorsque l'utilisateur passera la souris sur le paragraphe concerné.

Exemple 4 *Paragraphe*

```
1 | <p title="Titre du paragraphe"> Mon paragraphe</p>
```

1.6 Formatage du texte

Comme nous le verrons dans la section 2, le langage CSS offre de nombreuses possibilités de formatage de texte. Sans avoir recours au CSS, uniquement avec du code HTML, il est déjà possible de choisir quelques options de formatage basiques pour le texte.

Voici quelques marques utiles :

- `...` : pour mettre en gras (Bold)
- `<i>...</i>` : pour mettre en italique
- `<s>...</s>` : pour barrer du texte
- `... ` : pour définir du texte important
- `<u>...</u>` : pour souligner (Underline)
- `<mark>...</mark>` : pour surligner du texte en jaune

- `<h1>...</h1>` : pour appliquer un style de titre (*header*) de niveau 1. `<h2>`, `<h3>`, `<h4>` créent des titres plus petits (jusque **h6**). Les moteurs de recherche utilisent les titres pour structurer le contenu des pages web analysées. Ces balises doivent donc être utilisées pour les titres, pas uniquement pour rendre le texte plus grand ou écrire en gras.
- `<small>...</small>` : pour écrire en petit
- `<big>...</big>` : pour écrire en grand
- `_{...}` : pour écrire du texte en indice
- `^{...}` : pour écrire du texte en exposant.
- `<hr/>` : pour créer une ligne de séparation sur toute la largeur du document.
- `<details>... </details>` (non supporté par Internet Explorer) : pour spécifier des détails additionnels que l'utilisateur peut afficher ou cacher sur demande. Au sein de cet environnement, les balises `<summary>...</summary>` sont utilisées pour définir un titre visible par l'utilisateur. L'utilisateur pourra cliquer sur ce titre pour rendre les détails visibles.

Ces balises peuvent se combiner. On veillera toutefois à ne pas les croiser, c'est à dire que si une balise a été ouverte en dernier, c'est celle-là qu'on fermera en premier. Par exemple, on écrira

`<i> Un texte en gras et italique </i>`

et non pas `<i> Un texte en gras et italique </i>`.

Exemple 5 *Formatage de texte*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title> Police de caractères</title>
5  </head>
6  <body>
7    Un texte non formaté<br/>
8    <b> Un texte en gras</b><br/>
9    <i> Un texte en italique </i><br/>
10   <u> Un texte souligné</u><br/>
11   <mark> Un texte surligné</mark><br/>
12   <b><i> Un texte en gras et en italique</i></b><br/>
13   <h1> Titre 1</h1>
14   <h2> Titre 2</h2>
15   <h3> Titre 3</h3>
16   Une ligne <hr/>
17   <details>
18     <summary> Copyright 1999-2014. </summary>
19     All content and graphics on this web site are the property
20       of the company Refsnes Data.
21   </details>
22 </body>
    </html>

```

1.7 Caractères spéciaux

Certains caractères de notre alphabet pourraient ne pas être compris par certains navigateurs (du moins dans leurs versions anciennes), ou entrer en conflit avec les marques (quid du symbole *plus petit que* <) ou pas interprétés comme on le souhaiterait. Pour conserver une mise en page après un copier/coller d'un autre document au format quelconque avec un style "vieille machine à écrire" dans le navigateur, on utilise la marque `<pre>...</pre>`.

Les caractères spéciaux commencent par `&` et se terminent par `;`. Quelques exemples :

- ` ` ; : espace blanc
- `&` ; : symbole &
- `é` ; , `è` ; , `ê` ; : é, è, ê. Le premier "e" est remplacé par n'importe quelle lettre, par exemple pour obtenir un "à", on écrira `à`. Notons que ce codage rigoureux n'est plus nécessaire aujourd'hui.
- `<` ; , `>` ; : < et >

Exemple 6 Caractères spéciaux

```

1  <!DOCTYPE >
2  <html>
3  <head>
4    <title> Format </title>
5  </head>
6  <body>
7    <pre>
8      Ce texte apparaît tel qu'il a été introduit ,
9      sans ajouter de marques HTML
10   </pre>
11   <br />
12   Des caractères particuliers: <br/>
13   &nbsp; &amp;nbsp; <br/>
14   &eacute; &amp;eacute; <br/>
15   &egrave; &amp;egrave; <br/>
16   &ecirc; &amp;ecirc; <br/>
17   &lt; &amp;lt; <br/>
18   &gt; &amp;gt; <br/>
19 </body>
20 </html>

```

1.8 Liens

Pour associer un lien vers un autre document à un texte ou une image, on utilise la marque ` ... `. Ce qui se trouve entre ces deux marques (image ou texte) sera cliquable pour exécuter le lien. Par défaut, un lien non visité apparaît en bleu souligné, un lien visité en mauve et un lien actif en rouge.

L'adresse est une URL (*Uniform Resource Locator*). C'est un format d'adressage normalisé. L'URL la plus simple est le chemin d'accès à une autre page

Internet (protocole http). On pourra aussi utiliser d'autres protocoles dans une URL, par exemple pour envoyer un email (protocole `mailto:adresse_email`).

On peut ajouter un argument `target` à la marque `<a>` pour indiquer dans quelle fenêtre ouvrir le lien. En particulier, `target="_blank"` provoque l'affichage du lien dans un nouvel onglet du navigateur.

1.8.1 Ancres

Lorsque qu'un lien est défini vers une autre page, il pointe par défaut vers le sommet de cette page. Il est cependant possible de définir des *ancres* (des étiquettes) dans le document pour demander au navigateur d'afficher le document à cet endroit. La définition d'ancres peut également servir à accéder, lorsque la page est très longue, à un endroit précis de la page en cliquant simplement sur un lien donné en haut de page. Cela évite à l'internaute de chercher le seul paragraphe qui l'intéresse en faisant défiler l'entièreté de la page. On trouve aussi souvent des ancres permettant de remonter vers le haut de page par exemple.

La syntaxe de définition d'une ancre est : `` à ajouter dans le code à l'endroit où on veut que l'ancre pointe. Ensuite, pour créer le lien qui, lorsque l'internaute cliquera dessus, aura pour effet d'afficher la page à cet endroit précis (haut de page, bas de page, image, paragraphe,...), on ajoutera dans le lien le nom de l'ancre précédé du caractère `#` à la fin de l'URL. S'il s'agit d'une ancre positionnée sur la même page, on peut se contenter de `href="#nom_ancre"`.

Exemple 7 Liens externes et ancres

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title> Liens </title>
5  </head>
6  <body>
7    Ancre du haut <br/>
8    <a id="enhaut" />
9
10   Liens et email:<br />
11   <a href="http://www.sig.egss.ulg.ac.be"> Lien vers le site
12     SIG </a><br />
13   <a href="mailto:Bidule@ulg.ac.be"> Email </a><br />
14   <a href="html.htm"> Lien relatif </a><br />
15   <a href="liens.htm#anc1"> Lien vers l'ancre du bas </a>
16   <br /><br /><br /><br /><br /><br /><br /><br /><br />
17   <br /><br /><br /><br /><br /><br /><br /><br /><br />
18   <br /><br /><br /><br /><br /><br /><br /><br /><br />
19   <br /><br /><br /><br /><br /><br /><br /><br /><br />
20   <a id="anc1" /> L'ancre est placée ici <br />
21
22   <a href="#enhaut"> Retour en haut </a>
23 </body>
24 </html>

```

1.9 Images

Pour ajouter une image dans un document, on utilise la balise (vivant seule) ``. Ses paramètres sont :

- `src="chemin_d'accès_à_l'image"` : la source. Attention de ne pas spécifier une image qui n'est présente que sur votre disque, mais bien une image présente sur le serveur Internet. Ce paramètre est obligatoire.
- `alt="texte"` : un texte alternatif pour l'image (légende) dont l'intention est de remplacer l'image afin que le contenu de la page soit entièrement compréhensible par l'internaute même si les images ne peuvent être affichées. Ce paramètre est obligatoire.
- `width="pixels/pourcentage"` : la largeur de l'image en pixels (points de l'écran) ou en pourcentage de sa taille réelle. Un affichage classique d'un écran fait 800 pixels de large, mais cela peut être plus selon les ordinateurs. Si le paramètre n'est pas indiqué, la taille d'origine de l'image est utilisée.
- `height="pixels/pourcentage"` : la hauteur de l'image en pixels (points de l'écran) ou en pourcentage de sa taille réelle. Un affichage classique d'un écran fait 600 pixels de haut, mais cela peut être plus selon les ordinateurs.

Pour définir un environnement figure, la balise `` peut également être entourée des balises `<figure>... </figure>`. Cela permet de donner un titre à l'image qui sera affiché sur la page.

Il existe trois grands formats d'image reconnus sur Internet : .JPG, .GIF et .PNG. Selon l'image et le format, la taille du fichier (et donc la rapidité de transfert sur Internet) varie. GIF et PNG ont l'avantage d'autoriser des images avec un fond transparent. GIF était en théorie (même si c'est peu respecté) soumis à des droits d'auteur. PNG est la version gratuite de GIF.

Exemple 8 Insertion d'image

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title> Images</title>
5 </head>
6 <body>
7   4 formats d'extension possibles pour les images JPEG: <br/>
8    jpeg <br/>
9    jpg <br/>
10   JPG <br/>
11   JPEG <br/>
12
13  Autres types d'extension possibles: <br/>
14   GIF <br/>
15   PNG <br/>
16
17  Paramètres/Attributs utilisés:<br/>
18  

```

```

19 <br/>
20
21 Environnement figure: <br/>
22 <figure>
23    GIF <br/>
24   <figcaption> Titre de l'image </figcaption>
25 </figure>
26 </body>
27 </html>

```

1.10 Vidéo

Pour ajouter une vidéo dans une page web, on utilise les balises `<video>...</video>`. Dans cet environnement, la balise `<source />` est utilisée pour spécifier le chemin d'accès à la vidéo ainsi que le type d'extension. Cette balise doit impérativement contenir les paramètres `src="..."` pour le chemin d'accès, et `type="..."` pour le type de vidéo. Actuellement, trois types de vidéo sont supportés : MP4, WebM et Ogg. L'extension MP4 est cependant la seule reconnue par tous les navigateurs.

Si du texte est écrit entre les balises `<video>...</video>`, celui-ci ne sera affiché que si le navigateur ne reconnaît pas le type de vidéo ou ne supporte pas l'insertion de vidéos.

Plusieurs paramètres optionnels peuvent être précisés dans la balise `<video>` pour personnaliser l'affichage de la vidéo :

- `width="..."` et `height="..."` permettent de régler la taille de la fenêtre vidéo en pixels
- `autoplay` : pour que la vidéo démarre dès qu'elle est chargée
- `controls` : pour afficher un bouton play/pause sur la vidéo
- `loop` : pour que la vidéo tourne en boucle
- `muted` : pour couper le son de la vidéo
- `poster="..."` : pour définir le chemin d'accès à une image qui s'affichera tant que la vidéo charge ou que l'utilisateur n'a pas appuyé sur le bouton play.

Exemple 9 Insertion de vidéo

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title> Vidéo </title>
5 </head>
6 <body>
7   3 types de vidéos supportés: <br/>
8   <video autoplay loop poster="image.jpg">
9     <source src="movie.mp4" type="video/mp4"/> <br/>
10    <source src="movie.ogg" type="video/ogg"/> <br/>
11    <source src="movie.webm" type="video/webm"/> <br/>
12    Votre navigateur ne permet pas la lecture de la vidéo
13  </video>
14 </body>

```


15 | `</html>`

1.11 Bande son

Pour ajouter une bande son à une page web, on utilise les balises `<audio>...</audio>`. Dans cet environnement, la balise `<source />` est utilisée pour spécifier le chemin d'accès à la bande son ainsi que le type d'extension. Cette balise doit impérativement contenir les paramètres `src="..."` pour le chemin d'accès, et `type="..."` pour le type de fichier audio. Actuellement, trois types de fichier audio sont supportés : MP3, wav et Ogg. L'extension MP3 est cependant la seule reconnue par tous les navigateurs.

Si du texte est écrit entre les balises `<audio>...</audio>`, celui-ci ne sera affiché que si le navigateur ne reconnaît pas le type ne supporte pas l'insertion de bande son.

Plusieurs paramètres optionnels peuvent être précisés dans la balise `<audio>` pour personnaliser la lecture :

- `autoplay` : pour que la bande son démarre dès qu'elle est chargée
- `controls` : pour afficher un bouton play/pause
- `loop` : pour que la bande son tourne en boucle

Exemple 10 Insertion de fichier audio

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title> Fichier audio </title>
5  </head>
6  <body>
7    3 types de fichiers supportés: <br/>
8    <audio autoplay loop >
9      <source src="son.mp3" type="audio/mp3"/> <br/>
10     <source src="son.ogg" type="audio/ogg"/> <br/>
11     <source src="son.wav" type="audio/webm"/> <br/>
12     Votre navigateur ne permet pas la lecture du fichier sonore
13   </audio>
14 </body>
15 </html>

```

1.12 Page dans une page

Pour insérer un cadre permettant de naviguer sur une autre page au sein même de votre page web, on utilise les balises `<iframe src="lien">...</iframe>`. L'URL de la page à inclure est spécifié grâce au paramètre `src="..."`. Le texte placé entre ces deux balises sera affiché dans le cas où le navigateur ne supporterait pas l'utilisation de `iframe`.

```

1  <iframe src="demo_iframe.htm" height="200" width="300">
2    Votre navigateur ne peut lire ce lien
3  </iframe>

```

1.13 Tabulations et listes

Les tabulations n'existent pas en HTML. Pour créer des décalages, on peut utiliser des espaces blancs (), des listes ou des tables (cfr plus loin).

Pour créer une liste à puces, on utilise soit ... pour une liste numérotée (*Ordered List*) ou ... pour une liste à puces non numérotée (*Unordered List*). Chaque élément de la liste commence par (*List Item*) et se termine par . Chaque élément de liste commence sur une nouvelle ligne et il n'est donc pas nécessaire d'utiliser une marque
.

Exemple 11 Listes

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title> Listes </title>
5 </head>
6 <body>
7   Pas de tabulations!<br/>
8   &nbsp;&nbsp;&nbsp;&nbsp;  Mais la possibilité <br/>
9   &nbsp;&nbsp;&nbsp;&nbsp;&nbsp; d'utiliser un <br/>
10  &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; caractère "blanc". <br/>
11  <hr />
12  <ul>
13    <li> 1er élément d'une liste non ordonnée </li>
14    <li> 2ème élément </li>
15    <li> 3ème élément </li>
16  </ul>
17  <ol>
18    <li> 1er élément d'une liste ordonnée </li>
19    <li> 2ème élément </li>
20    <li> 3ème élément </li>
21  </ol>
22 </body>
23 </html>

```

Par défaut, les listes à puces (numérotées et non numérotées) vont afficher à l'écran un type particulier de puce. Il est toutefois possible de définir soi-même le type de puce que l'on souhaite voir apparaître. Pour ce faire, il suffit d'utiliser le paramètre `type` et de lui donner la valeur voulue. Il existe de nombreuses possibilités de personnalisation. Certains sites listent ces possibilités sur Internet. Voici quelques exemples de valeurs pouvant être utilisées : <ul type="square"> (ou "circle", "disk", ...) et <ol type="A"> (ou "a", "I", "1" ...). Les listes à puces n'utilisent que les puces de type "non-numéroté" et les listes à puces n'utilisent que les puces de type "numéroté".

Exemple 12 Types de puces

```

1 <ul type="square">
2   <li> ... </li>
3   <li> ... </li>
4 </ul>
5 <ol type="A">

```

```
6     <li> ... </li>
7     <li> ... </li>
8 </ul>
```

1.14 Les tables

Pour créer un tableau, l'entièreté du contenu du tableau doit être entouré des balises `<table>...</table>`. Un titre peut être spécifié directement après la balise d'ouverture du tableau par les balises `<caption>...</caption>`.

A l'intérieur des balises `<table>...</table>`, la définition des éléments du tableau se fait par ligne. Chaque nouvelle ligne commence par `<tr>` (*table row*) et se termine par `</tr>`. Au sein d'une ligne, la définition se fait ensuite par colonne : une colonne est délimitée par `<td>...</td>` (*table data*).

Exemple 13 *Tableau*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title> Tables </title>
5 </head>
6 <body>
7   <table>
8     <tr>
9       <td> (1,1) </td>
10      <td> (1,2) </td>
11      <td> (1,3) </td>
12    </tr>
13    <tr>
14      <td> (2,1) </td>
15      <td> (2,2) </td>
16      <td> (2,3) </td>
17    </tr>
18  </table>
19  <hr />
20
21 </body>
22 </html>
```

Remarque :

On prendra l'habitude *d'indenter* le code HTML, comme ci-dessus, c'est à dire d'utiliser des tabulations dans le code afin de rendre plus visible certains blocs de code et améliorer la lisibilité de celui-ci. Cela permet également une vérification plus aisée que chaque balise possède sa balise de fermeture correspondante.

Voici également quelques paramètres utiles de la balise `<td>`

- `colspan="nbre_entier"` : pour fusionner plusieurs colonnes ("cellules") d'un tableau ensemble.
- `rowspan="nbre_entier"` : pour fusionner plusieurs lignes d'un tableau ensemble au sein d'une cellule.

Exemple 14 *Colspan*

```
1 <table>
2   <tr>
3     <td colspan="2"> 2 colonnes fusionnées</td>
4   </tr>
5   <tr>
6     <td> 1ère colonne</td>
7     <td> 2ème colonne</td>
8   </tr>
9 </table>
```

2 Les feuilles de style CSS

Dans la section précédente, nous avons vu quelques balises HTML permettant de mettre en page et de formater le contenu d'une page web. Cependant, si on se restreint au HTML, les possibilités esthétiques sont limitées. Les sites Web de nos jours utilisent le langage CSS (pour Cascading Style Sheets) afin de définir et personnaliser la présentation de documents HTML et XHTML.

Grâce au langage CSS, il est possible de stocker toutes les informations concernant le style d'une page web, et même d'un site web entier, dans un seul fichier. Ce fichier, qu'on appelle *feuille de style* porte l'extension `.css`. L'avantage est qu'il est possible de changer l'apparence d'un site web entier, sans modifier son contenu, en changeant ce seul fichier `.css`.

2.1 Insertion de règles CSS

Il y a plusieurs façons d'appliquer des règles CSS à un document HTML, en fonction de leur portée. La première solution est d'écrire les règles CSS dans une feuille de style externe. De cette manière, les règles de style peuvent être appliquées à plusieurs pages web. L'apparence d'un site web entier peut alors être modifiée seulement en changeant ce fichier. Si des règles CSS ne doivent être appliquées qu'à une seule page du site, il est également possible de les insérer au sein même de la page HTML concernée, dans l'entête. Enfin, pour définir des règles de style pour une seule balise au sein d'une seule page, on peut également insérer ces règles directement au sein de la balise concernée.

2.1.1 Feuille de style externe

Une feuille de style externe est un fichier ne contenant que des règles de style CSS (aucune instruction HTML). Ce fichier peut être écrit à l'aide de n'importe quel éditeur de texte et doit être sauvé avec l'extension `.css`. Considérons la feuille de style suivante, dénommée `filestyle.css` dans la suite

Exemple 15 *Feuille de style externe*

```
1 .ital { font-style: italic; }
2 .center { text-align: center; }
3 h1 { color: red ; }
4 h2 { color: blue ; }
```

Chaque page HTML à laquelle ces règles doivent être appliquées doit contenir une référence à ce fichier `.css` dans son entête, c'est à dire, entre les balises `<head>...</head>`. Pour référencer une feuille de style, on utilise la balise `<link>` de la façon suivante :

```
1 <head>
2   <link rel="stylesheet" type="text/css" href="filestyle.css" />
3 </head>
```

Les titres `<h1>` et `<h2>` de ces pages seront respectivement affichés en rouge et en bleu, et les classes `"ital"` et `"center"` pourront être utilisées dans ces pages HTML comme précédemment.

2.1.2 Feuille de style interne

Lorsqu'on veut définir un style différent pour une seule page web, il est possible d'insérer les règles CSS directement dans le document HTML correspondant. Les règles doivent alors être définies dans l'entête (au sein des balises `<head>...</head>` et être entourées des balises `<style> ... </style>`. La portée de ces règles sera alors restreinte à cette seule page.

Exemple 16 *Page HTML avec feuille de style interne*

```
1 <head>
2   <style>
3     h1 { color: orange ; }
4   </style>
5 </head>
6 <body>
7   <h1> Un titre </h1>
8 </body>
```

Notez que cette méthode ne devrait être utilisée que si une règle ne doit être appliquée que dans une seule page. Dès qu'une règle s'applique à plus d'une page, on privilégiera la première solution puisqu'elle permet d'éviter les répétitions et de modifier l'apparence de plusieurs pages en ne changeant qu'un seul fichier.

Remarque importante. L'utilisation de feuilles de style externes peut être combinée avec les feuilles de style internes, en référençant dans l'entête du document HTML la feuille de style externe à l'aide des balises `<link>` et en ajoutant des règles internes au sein des balises `<style>...</style>`.

Exemple 17 *Feuilles de style externe et interne*

```
1 <head>
2   <link rel="stylesheet" type="text/css" href="filestyle.css" />
3   <style>
4     h1 { color: orange ; }
5   </style>
6 </head>
7 <body>
8   <h1> Un titre </h1>
9 </body>
```

Il se peut alors que plusieurs règles de style distinctes ait été définies pour le même élément. Par exemple, avec la feuille de style externe `filestyle.css`, le texte des balises `<h1>` devrait s'afficher en rouge. Néanmoins, la feuille de style interne contient également une autre règle de style pour les titres `<h1>` précisant que le texte devrait s'afficher en orange.

La règle qui s'appliquera sera toujours la dernière référencée dans le document HTML. Dans l'exemple 17, puisque la feuille de style interne est définie après la référence à la feuille de style externe, les éléments `<h1>` seront affichés en orange. Si par contre on référence la feuille de style externe après la feuille de style interne, comme dans l'exemple 18, les éléments `<h1>` seront affichés en rouge.

Exemple 18 *Feuilles de style externe et interne*

```
1 <head>
2   <style>
3     h1 { color: orange ; }
4   </style>
5   <link rel="stylesheet" type="text/css" href="filestyle.css" >
6 </head>
7 <body>
8   <h1> Un titre en rouge </h1>
9 </body>
```

2.1.3 **Inline**

Pour définir des règles de style ponctuellement pour une seule balise dans une seule page, on peut insérer ces règles directement au sein de la balise concernée à l'aide d'un paramètre `style="..."`. Les déclarations CSS doivent être entourées de guillemets et chaque déclaration se termine par un point-virgule.

Exemple 19 *Inline*

```
1 <h1 style="color:red; text-align:center;"> Un titre </h1>
```

Attention que cette alternative doit être utilisée avec parcimonie et uniquement pour définir un style de façon ponctuelle. Dès qu'une règle de style doit être utilisée pour plusieurs balises et/ou dans plusieurs pages, les feuilles de style interne et externe doivent être employées.

2.2 **Quelques déclarations utiles**

Ci-dessous sont listées quelques propriétés de style intéressantes. Une liste exhaustive peut être consultée à l'adresse incontournable <http://www.w3schools.com/css>.

2.2.1 **Formater du texte**

Plusieurs propriétés permettent de formater du texte

- `color` : permet de spécifier une couleur de texte
- `text-align` : permet de spécifier l'alignement du texte (`right`, `left` ou `center`)
- `font-size` : permet de spécifier la taille du texte
- `font-family` : permet de spécifier la police

Exemple 20 *Exemple de formatage de texte CSS*

```
1 p {
2   color: lightblue;
3   text-align: right;
4   font-size: 16px;
5   font-family: "Times New Roman";
6 }
```

2.2.2 Marges, bordures et padding

Les éléments HTML (tableaux, images, paragraphes, liens, sections, footer, header, ...) peuvent être vus comme des blocs. Ces blocs se composent de marges, de bords, de padding et du contenu de l'élément, comme l'illustre la figure 1.

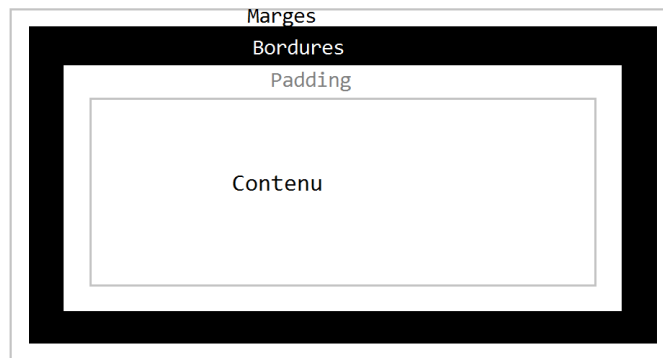


FIGURE 1 – Bloc HTML

Marges. Les propriétés concernant les marges permettent de générer de l'espace autour d'un élément, de le séparer des éléments environnants. Chaque marge peut être ajustée individuellement.

```
1 | p {
2 |     margin-top: 100px;
3 |     margin-right: 100px;
4 |     margin-bottom: 150px;
5 |     margin-left: 80px;
6 | }
```

Le raccourci suivant permet d'arriver au même résultat, pour autant que les tailles des marges soient spécifiées dans cet ordre (haut, droite, bas, gauche).

```
1 | p { margin: 100px 150px 100px 80px; }
```

Bordures. Plusieurs propriétés permettent de spécifier le style, la largeur et la couleur des bordures d'un élément

- **border-style** : précise le type de bordure (pointillés, traits, aucune,...). Des valeurs possibles sont `dotted`, `dashed`, `solid`, `double`, `none`, `hidden`.
- **border-width** : permet de spécifier la taille de la bordure, en pixels
- **border-color** : permet de spécifier la couleur de la bordure
- **border-radius** : permet définir des bordures plus ou moins arrondies

Exemple 21 *Formatage des bordures*


```
1 | p {  
2 |     border-style: solid;  
3 |     border-width: 2px;  
4 |     border-color: red;  
5 |     border-radius: 5px;  
6 | }
```

Les trois premières déclarations peuvent être regroupées en une seule.

```
1 | p {  
2 |     border: solid 2px red ;  
3 |     border-radius: 5px;  
4 | }
```

Padding. Les propriétés de padding permettent d'ajouter de l'espace entre la bordure d'un élément et son contenu. Chaque marge de padding peut être ajustée individuellement.

```
1 | p {  
2 |     padding-top: 10px;  
3 |     padding-right: 10px;  
4 |     padding-bottom: 50px;  
5 |     padding-left: 80px;  
6 | }
```

Le raccourci suivant permet d'arriver au même résultat, pour autant que les tailles soient spécifiées dans cet ordre (haut, droite, bas, gauche).

```
1 | p { padding: 10px 10px 50px 80px;}
```

Taille. La hauteur et la largeur d'un élément peuvent être définies à l'aide de règles CSS. Elles peuvent être spécifiées en pixels, en centimètres ou en pourcentage du bloc dans lequel se trouve l'élément. Attention que la hauteur et la largeur d'un élément n'incluent pas le padding, les bords ni les marges, ils fixent seulement la taille du contenu à l'intérieur du padding, des bords et des marges.

```
1 | img { height: 200px; width: 50%;}
```

2.2.3 Arrière plan

Plusieurs propriétés permettent de personnaliser l'arrière plan d'un élément (surface à l'intérieur des bordures) :

- **background-color** : permet de spécifier une couleur d'arrière plan

```
1 | body { background-color: lightblue;}
```

- **background-image** : permet de spécifier une image d'arrière plan

```
1 | body { background-image: url("paper.gif");}
```

- **background-repeat: no-repeat** : par défaut, si l'image d'arrière plan est trop petite, celle-ci sera répétée en mosaïque. Cette déclaration permet que l'image ne soit montrée qu'une seule fois.
- **opacity** permet de spécifier la transparence d'un élément, ou d'une image.

```
1 | img { opacity: 0.5; }
```

2.2.4 Tables

Les bordures des éléments d'une table, à savoir des éléments `<table>`, `<tr>` et `<td>` peuvent être ajustés avec la propriété **border** vue précédemment. Pour que les bords de chaque cellule de la table soient agglomérés entre eux (et non dédoublés), la déclaration **border-collapse: collapse;** peut être utilisée.

Exemple 22 Agglomérer les bords d'un tableau

```
1 | table, tr, td { border-collapse: collapse; }
```

Pour contrôler l'espace entre les bordures et le contenu des cellules, la propriété **padding** définie précédemment peut être appliquée aux éléments `<td>`. L'alignement horizontal du texte au sein des cellules se fait à l'aide de la propriété **text-align**. Pour l'alignement vertical du texte, on peut utiliser la propriété **vertical-align** avec comme valeur **top**, **bottom** ou **middle**. Avec les déclarations vues précédemment, on peut également définir un arrière plan pour chaque cellule.

2.2.5 Liens

Les règles de style précédentes (formatage de texte, arrière plan, marges, bordures,...) s'appliquent également aux liens `<a>`. La règle **text-decoration: none** permet de supprimer le soulignement par défaut.

De plus, le style d'un lien peut être défini en fonction de son état. Par exemple, on peut choisir d'afficher tous les liens en bleu sauf ceux qui ont déjà été visités qu'on affichera en gris. Les quatre états possibles sont

- **a:link** lien normal non visité
- **a:visited** lien que l'utilisateur a déjà visité
- **a:hover** un lien sur lequel l'utilisateur glisse la souris
- **a:active** un lien au moment où l'utilisateur clique dessus

```
1 | <style>
2 |   a:link    {color: green; background-color: transparent; text-
3 |     decoration: none}
4 |   a:visited {color: pink; background-color: transparent; text-
5 |     decoration: none}
6 |   a:hover   {color: red; background-color: transparent; text-
7 |     decoration: underline}
8 |   a:active  {color: yellow; background-color: transparent; text-
9 |     -decoration: underline}
10 | </style>
```

Attention. Les règles CSS pour les différents états d'un lien doivent impérativement être définis dans cet ordre.

Dans l'exemple ci-dessous, plusieurs règles CSS sont combinées pour afficher les liens comme des boutons cliquables. La déclaration `display: block;` permet d'afficher les liens comme des blocs dont la surface totale est cliquable.

Exemple 23 *Lien personnalisé*

```
1 | a {  
2 |     background-color: lightblue;  
3 |     padding: 14px 25px;  
4 |     text-align: center;  
5 |     text-decoration: none;  
6 |     display: block;  
7 |     margin: 10px;  
8 |     width: 100px;  
9 | }
```

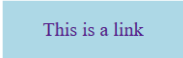


FIGURE 2 – Résultat de l'exemple 23

2.2.6 Listes

Les propriétés de style vues précédemment peuvent également être appliquées aux listes ``, `` et à leurs éléments ``. Par exemple, on peut définir un arrière-plan, changer la police, ...

Exemple 24 *Liste personnalisée*

```
1 | li {  
2 |     background-color: red;  
3 |     color: white;  
4 |     padding: 14px 25px;  
5 |     text-align: center;  
6 | }
```

Pour utiliser une image comme symbole apparaissant devant chaque élément d'une liste, on peut utiliser la déclaration `list-style-image: url("image.jpeg");` et l'appliquer aux balises `` ou ``.

Dans de nombreux sites actuels, les listes sont utilisées pour créer des menus latéraux ou horizontaux référençant les différentes parties du site. L'exemple 25 illustre une façon de combiner les propriétés de style des listes et des liens pour créer un menu de navigation interactif.

Exemple 25 *Menu*

```
1 <html>
2 <head>
3 <style>
4   ul {
5     list-style-type: none;
6     margin: 0;
7     padding: 0;
8     width: 200px;
9     background-color: lightgray;
10  }
11  a {
12    display: block;
13    color: #000;
14    padding: 8px 16px;
15    text-decoration: none;
16  }
17  a:hover {
18    background-color: gray;
19    color: white;
20  }
21  a.aktif {
22    background-color: green;
23    color: white;
24  }
25 </style>
26 </head>
27 <body>
28 <ul>
29   <li><a class="aktif" href="#home"> Home</a></li>
30   <li><a href="#news"> News</a></li>
31   <li><a href="#contact"> Contact</a></li>
32   <li><a href="#about"> About</a></li>
33 </ul>
34 </body>
35 </html>
```

Les listes `ul` n'ont pas de symbole devant leurs éléments, leur largeur est de 200px et l'arrière plan est gris clair. Les liens de cette page sont affichés en tant que blocs cliquables (grâce à la déclaration `display:block`). Ils ne sont pas soulignés et l'espacement entre le texte et le bord est personnalisé. Les liens sont écrits en noir. Les liens dont l'état est `hover` (l'utilisateur passe sa souris dessus) ont des règles de style spécifiques : leur arrière plan est gris foncé et le texte est écrit en blanc. Enfin, les liens dont la classe est `"aktif"` ont un arrière-plan vert et sont écrits en blanc.

3 Structurer une page avec le CSS

Les pages web de la plupart des sites actuels se composent souvent de plusieurs blocs (titre, pied de page, contenu, menu latéral, menu horizontal,...), chacun possédant son propre style (arrière-plan colorés, bords, marges, police, taille d'écriture) et dans lesquels on retrouve plusieurs éléments comme un titre, du texte, des images,...

Nous avons déjà vu précédemment que les balises `<header>...</header>`



FIGURE 3 – Résultat de l'exemple 25. Lorsque l'utilisateur passe la souris sur un lien, celui-ci devient gris-foncé. Le lien actif a un arrière plan vert.

et `<footer>...</footer>` pouvaient être employées pour définir des blocs de titre et de pied de page, ou encore de titre de section et de pied de section.

Pour regrouper des éléments au sein d'un même bloc, les balises HTML `<div>...</div>` peuvent également être utilisées.

Exemple 26 *Bloc div*

```
1 <div>
2   <h1> Un titre </h1>
3   <p> Du texte </p>
4 </div>
```

L'utilisation de tels blocs a pour avantage qu'il est alors possible d'appliquer un style à l'ensemble du contenu d'un bloc `<div>` à l'aide de règles CSS, par exemple, uniformiser la police et la couleur du texte au sein d'un même bloc, définir un arrière plan derrière plusieurs éléments. En effet, comme les autres balises HTML, les balises `<div>` peuvent avoir des paramètres `class="..."` et `id="..."`. Il est donc possible de leur appliquer des règles de style définies dans une feuille de style externe ou interne. Si des règles de style ne s'appliquent qu'à un seul bloc dans une seule page, on peut aussi utiliser des règles inline. L'exemple ci-dessous définit deux blocs ayant des styles différents.

Exemple 27 *Appliquer un style à un bloc div*

```
1 <div style="background-color:gray;color:white;padding:20px;">
2   <h2> Un titre </h2>
3   <p> Du texte</p>
4 </div>
5 <br/>
6 <div style="background-color:black;color:white;padding:20px;">
7   <h2> Un autre titre </h2>
8   <p> Du texte</p>
9 </div>
```

Les blocs définis à l'aide de balises `<div>` commencent toujours à une nouvelle ligne et prennent la largeur maximale disponible. Pour au contraire, créer des blocs au sein même d'un texte sans passer à la ligne et en n'utilisant pas forcément toute la largeur nécessaire, on utilisera plutôt les balises `...`.

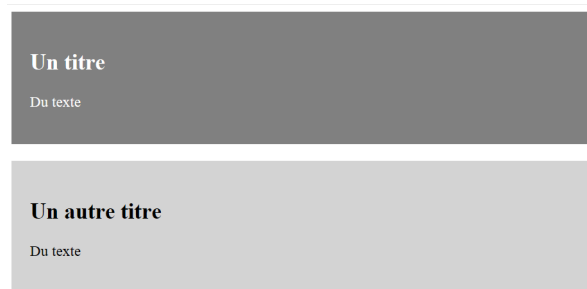


FIGURE 4 – Résultat de l'exemple 27

Ces balises peuvent également avoir des paramètres `class="..."` et `id="..."`. Les balises `` permettent de définir un style uniquement pour certaines parties en plein milieu d'un texte.

Exemple 28 *Bloc span*

```
1 | Une promotion <span style="background-color:blue; color:white;
   | font-size:150%;" > importante </span> à ne pas manquer
```

Une promotion **importante** à ne pas manquer

FIGURE 5 – Résultat de l'exemple 28

3.1 Éléments flottants

Pour structurer les éléments d'une page web et contrôler la position de différents blocs, il faut leur appliquer des règles de style spécifiques. La propriété `float` permet de définir un élément (bloc, paragraphe, image,...) flottant. Les éléments définis après un tel élément flottent autour de celui-ci. Dans l'exemple de règle CSS suivant, on définit une classe `floating` d'images flottantes alignées à droite.

Exemple 29 *Image flottante*

```
1 | img.floating { float: right; }
```

Une image ayant pour paramètre `class="floating"` s'affichera comme dans la figure 6. Le texte écrit après l'insertion de cette image flotte autour de celle-ci.

Pour justement éviter que les éléments après une image flottante ne l'entourent, la propriété `clear` peut être utilisée. Appliquée à un élément (paragraphe, bloc,...), la déclaration `clear:left;` spécifie qu'aucun autre élément flottant ne pourra se trouver à gauche de cet élément.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor.

Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui eget tellus gravida venenatis. Integer fringilla congue eros non fermentum. Sed dapibus pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.



FIGURE 6 – Résultat de l'exemple 29

Exemple 30 *Propriété clear*

```

1 <html>
2   <head>
3     <style>
4       img { float: right;}
5       .div1 { border: 1px solid red;}
6       .div2 { border: 1px solid red; clear:right;}
7     </style>
8   </head>
9   <body>
10
11  <h2> Sans clear</h2>
12  
13  <div class="div1"> div1 - L'élément div1 est défini après l'
    image flottante dans le code HTML. Puisque l'image est
    flottante, le texte du bloc div1 flotte autour de l'image </
    div>
14
15  <h2> Avec clear</h2>
16  
17  <div class="div2"> div2 - En spécifiant clear:right, aucun
    élément ne peut flotter à droite du bloc div2.</div>
18 </body>

```

3.2 Overflow

Nous avons vu qu'il était possible de fixer la hauteur et la largeur des éléments à l'aide des propriétés `height` et `width`. Que se passe-t-il cependant si la longueur ou la largeur du contenu dépasse les tailles fixées ? Si aucune précision n'est faite, le contenu sortira du cadre prévu. La déclaration `overflow:auto` permet d'ajouter automatiquement des barres déroulantes lorsque le contenu est trop grand. Cette propriété est illustrée dans l'exemple 31.

Sans clear

div1 - L'élément div1 est défini après l'image flottante dans le code HTML. Puisque l'image est flottante, le texte du bloc div1 flotte autour de l'image



Avec clear

div2 - En spécifiant clear:right, aucun élément ne peut flotter à droite du bloc div2.



FIGURE 7 – Résultat de l'exemple 30

3.3 Exemple de structure de page

Voici un exemple de structure de page qu'il est possible de réaliser avec les propriétés CSS vues précédemment. Afin de bien comprendre la signification de chacune des déclarations, nous vous invitons à tester par vous-même l'effet de la suppression de chacune d'entre-elles.

Exemple 31 *Structure de page*

```

1 <html>
2 <head>
3 <style>
4   header, footer {
5     padding: 16px;
6     color: black;
7     background-color: green;
8     clear: left;
9     text-align: center;
10    border-radius: 20px;
11  }
12  div.nav {
13    float: left;
14    max-width: 160px;
15    margin: 0;
16    padding: 16px;
17  }
18  div.content {
19    margin-left: 170px;
20    background-color: lightgray;
21    border-radius : 20px;
22    padding: 1em;
23    height: 200px;
24    overflow: auto;
25  }
26  ul {
27    list-style-type: none;
28    margin: 0;
29    padding: 0;
30    width: 150 px;

```



```

31     background-color: lightgray;
32 }
33 a {
34     display: block;
35     color: #000;
36     padding: 8px 16px;
37     text-decoration: none;
38     font-family: "Calibri";
39     font-size: 150%;
40 }
41 a:hover {
42     background-color: gray;
43     color: white;
44 }
45 a.actif {
46     background-color: green;
47     color: white;
48 }
49 img {
50     float:right;
51     border-radius: 5px;
52     margin-left: 5px;
53 }
54 </style>
55 </head>
56 <body>
57
58 <div class="container">
59
60 <header>
61     <h1>Image Gallery</h1>
62 </header>
63
64 <div class="nav">
65     <ul class="ref">
66         <li><a href="#" class="actif"> <div class="button" > Forest
67         </div> </a></li>
68         <li><a href="#"> <div class="button" > Beach </div> </a></li>
69         <li><a href="#"> <div class="button" > City </div> </a></li>
70     </ul>
71 </div>
72
73 <div class="content">
74     <h1>Forest</h1>
75     
77     <p> Ce texte entoure la photo car tout ce qui est placé après
78     un élément flottant flotte autour de celui-ci pour autant qu'
79     'on n'ait pas précisé la propriété clear. Ce texte entoure
80     la photo car tout ce qui est placé après un élément flottant
81     flotte autour de celui-ci pour autant qu'on n'ait pas
82     précisé la propriété clear. Ce texte entoure la photo car
83     tout ce qui est placé après un élément flottant flotte
84     autour de celui-ci pour autant qu'on n'ait pas précisé la
85     propriété clear. </p>
86 </div>

```

```

77 <footer> Copyright &copy; ... </footer>
78 </div>
79 </body>
80 </html>

```

L'en tête et le pied de page (balises `<header>` et `<footer>`) ont le même style : le texte est écrit en noir sur un arrière plan vert et le texte est aligné au centre. La déclaration `clear:left` signifie que rien ne peut flotter à gauche de ces éléments. Les bordures sont arrondies.

Le bloc ayant pour classe `nav` est flottant et aligné à droite. Sa largeur maximale est de 160px. Le bloc ayant pour classe `content` a une marge de 170 px à gauche, un arrière plan gris clair et des bords arrondis. Sa hauteur est fixée à 200px. Si le contenu d'un tel bloc dépasse cette hauteur, une barre de défilement est automatiquement ajoutée grâce à la déclaration `overflow:auto`;

Le menu latéral est créé avec les mêmes déclarations que dans l'exemple 25.

Enfin, les images de cette page sont alignées à droite et flottantes. Une marge de 10px les entoure à gauche.



FIGURE 8 – Résultat de l'exemple 31

4 Les formulaires HTML

Pour créer de l'interaction avec l'internaute et lui permettre d'introduire des informations dans une page, il faut créer un formulaire. Un formulaire permet la création d'objets graphiques interactifs et l'appel d'une procédure pour traiter a posteriori les informations introduites. Il permet de transmettre des informations au serveur. La syntaxe est la suivante :

Exemple 32 *Syntaxe de base d'un formulaire*

```
1 <form name="un_nom" action="la_procedure" method="la_méthode">
2 ...
3 </form>
```

Détaillons les paramètres de la balise `<form>` :

- `name="un_nom"` : le nom du formulaire, choisi librement.
- `action="la_procedure"` : la procédure est un programme défini ailleurs (soit écrit par vous-même, soit un programme existant) qui doit exister indépendamment. Cette procédure est un fichier contenant du code permettant d'utiliser les données entrées par l'utilisateur dans le formulaire. Par exemple, une procédure bien connue est un programme permettant d'envoyer par email le contenu d'un formulaire à un destinataire quelconque. Sur le serveur Internet de l'Ulg, ce programme se dénomme "getcomments.pl". Sa description complète se trouve à l'URL http://www.ulg.ac.be/segi/infoweb/segi_server/cgi/getcomments.html. Si c'est ce programme qu'on veut employer, on écrira `action="http://www.ulg.ac.be/cgi-bin/getcomments.pl"`. Nous verrons cette année comment créer de tels programmes.
- `method="la_méthode"` : la méthode dépend de l'action. C'est soit POST, soit GET.
 1. Méthode GET : lorsque le formulaire est envoyé, une URL est créée en utilisant la valeur de l'action et les données du formulaire, c'est à dire que les valeurs entrées par l'utilisateur dans le formulaire apparaissent dans l'URL. Les données entrées par l'internaute restent dans l'historique du navigateur. Par conséquent, il est conseillé de ne pas utiliser cette méthode pour des données confidentielles.
Notons que le fait que les données apparaissent dans l'URL implique que celles-ci ne peuvent pas dépasser une longueur maximum (on évitera donc cette méthode lorsque l'internaute doit soumettre un texte important dans un formulaire par exemple). La longueur maximale d'une URL est en effet de 2048 caractères.
 2. Méthode POST : les données sont envoyées au serveur sans les afficher dans l'URL. Puisque les données ne sont pas retenues dans l'historique de navigation, un rafraîchissement de la page implique que les

données doivent être re-soumises par l'internaute. Cette méthode est donc plus sécurisée. Pour cette méthode, il n'y a pas de restrictions de taille des données.

D'autres paramètres facultatifs permettent de personnaliser le formulaire, comme le paramètre `autocomplete="on"` servant à spécifier si le navigateur suggère automatiquement des suggestions basées sur des données précédemment entrées par l'utilisateur.

4.1 Objets d'un formulaire

Un formulaire est composé de différents objets `<input>` (les cases du formulaire), chacun portant un nom et un type. Chaque objet est l'équivalent d'une variable. Le nom de la variable est défini par le paramètre `name` et son type par le paramètre `type`. La syntaxe est la suivante

`<input type="un_type" name="un_nom" />`. Le nom donné à l'input sera important pour identifier les données lors de leur récupération. On évitera donc de donner des noms identiques à des inputs différents !

Selon le type d'input, l'utilisateur doit entrer des données de type différents. Détaillons les types principaux d'input :

Texte. `type="text"` pour obtenir une case permettant l'introduction d'un texte. La taille de la case peut être modifiée grâce au paramètre `size="valeur"`. Il est possible de définir une valeur par défaut avec le paramètre `value="valeur"`. Dans ce cas, la case sera pré-remplie. Il est possible de spécifier à l'internaute une brève description du format attendu avec le paramètre `placeholder`.

Mot de passe. `type="password"` pour une case permettant l'introduction d'un texte caché au fur et à mesure de l'encodage par l'internaute. La taille maximale peut être spécifiée avec le paramètre `maxlength="valeur"`.

Nombre. `type="number"` pour obtenir une case permettant l'introduction d'un nombre. Les valeurs minimales et maximales peuvent être fixées à l'aide des paramètres `min="valeur"` et `max="valeur"`. Le paramètre `step="valeur"` permet de restreindre les réponses acceptées à des valeurs équi-distances. Par exemple, `input type="number" min="0" max="100" step="10"/>` permet d'entrer un nombre de 0 à 100 par pas de 10.

Date. `type="date"` pour obtenir une case permettant l'introduction d'une date. Les valeurs minimales et maximales peuvent être fixées à l'aide des paramètres `min="valeur"` et `max="valeur"`. En fonction du navigateur utilisé, un calendrier peut apparaître permettant à l'utilisateur de choisir une date.

Heure `type="time"` pour obtenir une case permettant l'introduction d'une heure.

Couleur `type="color"` pour une case permettant le choix d'une couleur. Il est possible de déterminer une couleur par défaut à l'aide du paramètre `value="color"`. Dans ce cas, la couleur spécifiée doit soit correspondre à une couleur prédéfinie dans en HTML (voir http://www.w3schools.com/colors/colors_names.asp, soit être spécifiée à l'aide du code HTML correspondant (voir http://www.w3schools.com/colors/colors_picker.asp).

Case à cocher. `type="checkbox"` : une case à cocher ou non. Toutes les cases à cocher doivent avoir des `name` différents. Il est possible de cocher une case par défaut en ajoutant le paramètre `checked`.

Bouton radio. `type="radio"` pour obtenir un bouton radio, c'est à dire une réponse unique à cocher parmi plusieurs. La différence avec le type `checkbox` décrit précédemment est que le bouton radio requiert de l'internaute qu'il fasse **un seul** choix (une seule case peut être cochée à la fois). Il y a un input de type radio pour chaque choix possible. Ce choix d'input est particulièrement adéquat pour demander le sexe de l'internaute par exemple. Lors de l'utilisation d'un bouton radio, il faut faire attention à plusieurs choses :

1. tous les boutons radio d'un même groupe doivent porter le même nom (attribut `name` identique, par exemple `name="sexe"`).
2. on ajoute un paramètre `value` différent pour chaque bouton radio afin de pouvoir identifier la valeur choisie par l'utilisateur lors du traitement des données (l'internaute est-il un homme ? `value="H"` ou une femme ? `value="F"`).

Un exemple est donné ci-dessous. Il est possible de cocher un bouton radio par défaut en ajoutant le paramètre `checked` à l'input correspondant.

Bouton soumettre. `type="submit"` : un bouton qui une fois pressé démarre l'exécution du programme spécifié dans le paramètre `action` du formulaire. Le texte affiché sur le bouton est spécifié par le paramètre `value`.

Bouton reset. `reset` : un bouton qui une fois pressé réinitialise le formulaire. Le texte affiché sur le bouton est spécifié par le paramètre `value`.

Données cachées. `type="hidden"` : une variable texte qui est cachée à l'utilisateur. Sa valeur est fixée par le paramètre `value`. Certains programmes de traitement d'informations requièrent des données spécifiques de type `hidden`, d'autres non. Ce type est par exemple indispensable pour définir des variables requises par le programme `"getcomments.pl"`, mais qui ne sont pas du ressort de l'utilisateur.

D'autres types d'input existent. On peut par exemple demander à l'utilisateur de joindre un fichier avec le type `file` tandis que le type `button` définit un bouton cliquable. Une liste complète des types d'inputs ainsi que de leurs attributs peut être trouvée à l'adresse <http://www.w3.org/community/webed/wiki/HTML/Elements/input>.

D'autres paramètres peuvent être ajoutés pour personnaliser les inputs :

- `required=1` pour indiquer qu'une entrée est requise pour cet input avant de pouvoir valider le formulaire
- `readonly=1` pour spécifier que le champ peut uniquement être lu et non édité
- `autocomplete='on'` pour que le navigateur suggère automatiquement des valeurs déjà entrées par l'utilisateur. Il est possible de spécifier `autocomplete='on'` pour le formulaire complet (dans la balise `<form>`) et `off` pour certains inputs et vice-versa.
- `autofocus` : pour que le curseur se place automatiquement dans la case correspondante à l'ouverture de la page.

Autres objets. Deux autres objets d'un formulaire utilisent une syntaxe différente de celle des inputs :

- `<textarea name="un_nom" rows="nb_ligne" cols="nb_carac">...</textarea>` : pour définir une case pouvant contenir un texte sur plusieurs lignes. Le paramètre `cols` indique le nombre de caractères maximum par ligne et `rows` indique le nombre de lignes.
- `<select name="un nom">...</select>` : pour créer un menu déroulant. Le paramètre `size` indique le nombre d'options montrées à l'internaute à la fois. Au sein de ces balises, chaque élément du menu est entouré des balises `<option>...</option>`. le paramètre `value` doit être ajouté à chaque option pour pouvoir récupérer ce que l'utilisateur a choisi. Le paramètre `selected` peut être ajouté à chaque option pour que celle-ci soit sélectionnée par défaut.

Le paramètre `required` peut également être utilisé pour ces objets.

Exemple 33 *Exemple de formulaire dont les données sont traitées par le programme "getcomments.pl"*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title> Formulaire </title>
5 </head>
6 <body>
7   <form name="formulaire"
8     action="http://www.ulg.ac.be/cgi-bin/getcomments.pl"
9     method="POST">
10     Texte:
11     <input name="LeTexte" type="text" size="20" value="valeur
      par défaut"/><br/>

```

```

12
13     Zone de texte:
14     <textarea name="LaZone" rows="3" cols="20">
15     valeur par défaut
16     </textarea><br/>
17     <hr/>
18
19     Un seul choix possible grâce aux boutons radio:<br/>
20     <input name="BoutonsRadio" type="radio" value="a"/> A <br/>
21     <input name="BoutonsRadio" type="radio" value="b" checked="
22         checked"/> B <br/>
23     <input name="BoutonsRadio" type="radio" value="c"/> C <br/>
24     <input name="BoutonsRadio" type="radio" value="d"/> D <br/>
25     <hr/>
26
27     Plusieurs choix possibles grâce aux cases à cocher:<br/>
28     <input name="Case_1" type="checkbox" checked="checked"/>
29     Case à cocher 1 <br/>
30     <input name="Case_2" type="checkbox"/> Case à cocher 2 <br
31     />
32     <input name="Case_3" type="checkbox" checked="checked"/>
33     Case à cocher 3 <br/>
34     <hr/>
35
36     Un menu déroulant:<br />
37     <select name="MonMenu">
38         <option value="menu1" > Menu 1 </option>
39         <option value="menu2" > Menu 2 </option>
40         <option value="menu3" selected="selected"> Menu 3 </
41         option>
42         <option value="menu4" > Menu 4 </option>
43     </select>
44     <hr />
45
46     Un menu déroulant de "type" liste:<br/>
47     <select name="MaListe" size="3">
48         <option value="elem1" > Elément 1 </option>
49         <option value="elem2" selected="selected"> Elément 2 </
50         option>
51         <option value="elem3" > Elément 3 </option>
52         <option value="elem4" > Elément 4 </option>
53     </select>
54     <hr />
55
56     Informations supplémentaires nécessaires par le programme:
57     <br/>
58     <input type="hidden" name="_returndocurl"
59     value="http://www.sig.egss.ulg.ac.be/contactack.htm"/>
60
61     Origine:
62     <input type="text" name="_mailfrom" size="40" value="@ulg.
63     ac.be"/><br/>
64
65     Destinataire:
66     <input type="text" name="_mailto" value="@ulg.ac.be"/><br/>
67
68     Sujet:

```

```

62     <input type="text" name="_mailsubj" value="Question WWW SIG
        "/><br/>
63
64     <input type="hidden" name="_mail_vals" value="all"/>
65     <input type="submit" value="Envoyer"/>
66     <input type="reset" value="Annuler"/><br/>
67 </form>
68 </body>
69 </html>

```

Notons que les `inputs` de type `hidden` sont nécessaires pour l'action définie par le programme "getcomments.pl". Néanmoins, lorsque vous écrirez vos propres programmes, ce ne sera plus nécessairement le cas : il s'agit ici de données spécifiques à l'action précisée.

Le résultat de ce code est montré dans la figure 9 :

Exemple 34 Exemple de formulaire

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title> Formulaire </title>
5  </head>
6  <body>
7    <form name="formulaire"
8      action="mapage.php"
9      method="GET">
10     Mot de passe:
11     <input name="mdp" type="password" size="8" /><br/>
12     Date:
13     <input type="date" name="bday" min="2000-01-02">?<br/>
14     Couleur:
15     <input type="color" name="favcolor" value="#ff0000"><br/>
16     <input type="submit" value="Envoyer"/>
17     <input type="reset" value="Annuler"/><br/>
18   </form>
19 </body>
20 </html>

```

Les cases d'un formulaire peuvent être regroupées en sections entourées de balises `<fieldset> ... </fieldset>`. Un titre peut être donné à chacune des sections entre les balises `<legend>...</legend>`.

Exemple 35 Regroupement en sections

```

1  <form>
2    <fieldset>
3      <legend> Personnalité:</legend>
4      Name: <input type="text"><br>
5      Email: <input type="text"><br>
6      Date of birth: <input type="date">
7    </fieldset>
8  </form>

```


Formulaire - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

file:///C:/Users/ULg/Desktop/Formulai Google

Formulaire +

Texte: valeur par défaut

valeur par défaut

Zone de texte:

Un seul choix possible grâce aux boutons radio:

☐ A

☒ B

☐ C

☐ D

Plusieurs choix possibles grâce aux cases à cocher:

☒ Case à cocher 1

☐ Case à cocher 2

☒ Case à cocher 3

Un menu déroulant:

Menu 3 ▼

Un menu déroulant de "type" liste:

Elément 1 ▲

Elément 2

Elément 3 ▼

Informations supplémentaires nécessaires par le programmes:

Origine: @ulg.ac.be

Destinataire: @ulg.ac.be

Sujet: Question WWW SIG

Envoyer Annuler

FIGURE 9 – Formulaire - résultat

Deuxième partie

Systèmes d'information dynamiques

Les sites web créés en utilisant uniquement du code HTML et des feuilles de style sont dits "statiques". En effet, ces pages peuvent présenter tous types de contenus, vidéos, sons, images,... mais elles sont toujours présentées de la même façon et ne changent pas en fonction des choix de l'internaute.

Par opposition aux pages statiques, les pages dynamiques utilisent les informations de l'internaute et permettent d'interagir avec celui-ci. Pensez aux sites sur lesquels vous pouvez vous inscrire, vous connecter, qui mémorisent votre nom, prénom, et vos informations personnelles. L'interaction avec l'internaute peut aller du simple affichage de son nom et prénom en haut d'une page jusqu'au stockage de ses informations dans une base de données complète, en passant par des promotions personnalisées, des cadeaux proposés en fonction du nombre d'achats effectués,... Nous avons vus précédemment comment afficher un formulaire permettant à l'internaute de rentrer un certain nombre de données. Cependant, nous ne savons pas encore comment récupérer et utiliser ces données pour rendre web interactifs.

Dans cette partie, nous allons utiliser le langage PHP pour créer des pages dynamiques. La première section explique les bases du langage PHP et entre autres, la récupération des données de formulaires HTML et la création de variables de session et de cookies. La seconde partie explique comment stocker et récupérer les données des clients dans une base de données.

5 Les bases du langage PHP

5.1 Principes et fonctionnement

Le PHP est un langage "coté serveur" qui permet de créer dynamiquement (instantanément à la demande d'un internaute) des pages Web (au format HTML ou XHTML). Le PHP est un langage extrêmement populaire, assez puissant pour être par exemple au coeur de la plus grande installation de blogs du web (WordPress), assez profond pour exécuter le plus grand réseau social (Facebook). C'est également un langage assez facile pour être le premier langage niveau serveur qu'un débutant peut apprendre.

Pourquoi le PHP ? Ce langage est supporté par de nombreuses plateformes et est compatible avec à peu près tous les serveurs utilisés de nos jours. Il permet de générer des pages au contenu dynamique, de récupérer les données de formulaires, d'envoyer et de recevoir des cookies, de manipuler des bases de données,...

PHP signifie "*PhP : Hypertext Preprocessor*". Son nom le définit parfaitement. Le fichier `.php` est en réalité un document html (ou xhtml) dans lequel

on a inséré à différents endroits des instructions du langage PHP. Tout fichier contenant du code PHP doit avoir l'extension `.php` pour être interprété au niveau du serveur web. Le serveur a pour mission d'analyser ce fichier, de repérer les endroits où du code PHP a été ajouté, d'exécuter ce code et de remplacer ce code par le résultat de l'exécution avant de transmettre le tout au navigateur de l'internaute. Le document web transmis au navigateur devant être en HTML, cela signifie que le résultat de l'exécution du code PHP ne peut générer que du texte en HTML (ou rien du tout ; ce qui est aussi du HTML).

A la différence du HTML, votre code n'apparaît pas dans la source de la page web, seules les informations qui résultent de son exécution par le serveur seront visible pour les visiteurs.

5.2 Structure de base

Un bloc de code PHP correctement écrit commence par `<?php` et se termine par `?>`. Chaque instruction à interpréter doit être suivie par un point-virgule. Nous obtenons donc la syntaxe suivante :

Exemple 36 *Structure de base d'un bloc php*

```
1 <?php
2     Instruction1;
3     Instruction2;
4     Instruction3;
5 ?>
```

Des blocs PHP peuvent être introduits à n'importe quel endroit du code HTML.

5.2.1 Commentaires

Il est possible d'écrire des commentaires au sein d'un bloc PHP également. Il existe plusieurs syntaxes permettant de définir des commentaires. Un commentaire d'une seule ligne débute par les symboles `//` tandis qu'un commentaire multi-lignes débute par `/*` et se termine par `*/`.

5.3 Fonction ECHO

Considérons tout d'abord une fonction de base qui nous permettra de mieux comprendre le fonctionnement du PHP. La première fonction que nous étudierons est la fonction `echo` qui est en fait la fonction d'écriture à l'écran. La syntaxe de cette fonction est la suivante :

```
echo " ce que vous désirez inscrire à l'écran " ;
```

Exemple 37 *Fonction echo*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title> Fonction echre </title>
```

```
5 </head>
6 <body>
7   <?php
8     echo "Salut la compagnie!";
9   ?>
10 </body>
11 </html>
```

5.4 Les variables

5.4.1 Le langage binaire

Avant de parler de variables, nous allons faire une petite parenthèse sur le fonctionnement d'un ordinateur. Un ordinateur ne parle en fait qu'une seule langue : le *langage binaire*, c'est à dire une simple suite de 0 et de 1. Il serait très difficile d'écrire des programmes informatiques en langage binaire. Les langages de programmation ont par conséquent été créés afin que nous ayons à disposition des instructions claires pour écrire nos programmes. Ces programmes seront ensuite compilés pour que ces instructions soient compréhensibles par nos machines.

Le *langage binaire* est une suite de 0 et de 1. Un tel chiffre est appelé *bit*. Un *octet* est un regroupement de 8 bits, un *Kilo octet* est un regroupement de 1024 octets, un *Méga octet* est un regroupement de 1024 ko, un *Giga octet* est un regroupement de 1024 Mo et un *Tera octet* est un regroupement de 1024 Go.

Dans la vie de tous les jours, nous avons l'habitude de compter en base 10, avec dix chiffres de 0 à 9. Ainsi, le nombre 123 signifie en fait :

$$1 * 10^2 + 2 * 10^1 + 3 * 10^0$$

Nous additionnons donc des puissances de la base utilisée (ici 10), multipliées par un nombre compris entre 0 et cette base moins 1 (ici, 9).

Dans le langage binaire, les seuls chiffres utilisés sont le 0 et le 1. Dès lors, comment stocker dans la mémoire d'un ordinateur la valeur correspondant à notre nombre 123 usuel ? Nous allons en fait écrire en base 2. Comme en base 10, l'idée est de décomposer ce nombre en une somme de puissances de la base (ici 2), multipliées par 0 ou 1. Puisque 2^6 est la plus grande puissance de 2 contenue dans le nombre 123, son écriture en base binaire contiendra donc 7 bits (un bit pour chaque puissance, dont la puissance zéro). En base binaire, la valeur 123 s'écrit donc

$$1111011$$

puisque

$$123 = 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$$

En PHP, des données peuvent être stockées dans des variables. Les variables sont des "containers" d'informations. C'est sous le langage binaire que ces informations seront stockées. Ces variables pourront être de plusieurs types, chacun

pouvant contenir un nombre spécifique de bits. Le type d'une variable sert à déterminer la façon de lire la suite de 0 et de 1 en mémoire (par exemple, *y a-t-il des décimales ou s'agit-il d'un nombre entier ?*).

5.4.2 Définition de variables

Dans le langage PHP, il n'est pas nécessaire de déclarer des variables avant de les utiliser. Une variable est automatiquement créée lorsqu'une valeur lui est affectée pour la première fois. Les variables peuvent avoir différents types : integer, string, boolean, float,... En PHP, le type d'une variable est déterminé par la valeur qui lui est affectée. On identifie une variable à un mot précédé de \$. Pour affecter une valeur à une variable, il suffit d'utiliser la syntaxe suivante : `$nom_variable = valeur_variable;`

Exemple 38 Variables

```
1 <?php
2 $varBool = false;
3 $varReel = 23.4;
4 $varTexte = "Salut la compagnie!";
5 ?>
```

Lorsqu'on assigne une chaîne de caractères à une variable, il faut entourer cette chaîne de guillemets ou d'apostrophes. Ceci est détaillé dans la prochaine section.

Il y a certaines règles à respecter pour le nom donné aux variables :

- le nom d'une variable commence toujours par \$
- le nom des variables peut être court `$x` ou plus explicite `$volume_total`.
- le nom d'une variable ne peut commencer que par une lettre ou un `_` (pas par un nombre)
- le nom d'une variable ne peut contenir que des caractères alphanumériques ou des `_`.
- le nom d'une variable est sensible à la casse : les noms `$age` et `$Age` ne désignent pas les mêmes variables

5.5 Chaînes de caractères

Pour affecter du texte à une variable ou manipuler ce texte dans des instructions, il faut le délimiter avec soit des guillemets, soit des apostrophes. Le résultat n'est pas tout à fait identique. Si le texte entre guillemets contient le nom d'une variable, cette variable sera remplacée par sa valeur dans le texte résultant. Par contre, lorsque le texte est encadré par des apostrophes, il n'y a pas de traitement du texte et il apparaît exactement comme indiqué entre les apostrophes.

Exemple 39 Guillemets et apostrophes - chaîne de caractères

```
1 <?php
2     $nom = "visiteur";
3     echo "Bonjour $nom ";
4 ?>
```

Le code affichera à l'écran : *Bonjour visiteur*

```
1 <?php
2     $nom = "visiteur";
3     echo 'Bonjour $nom ';
4 ?>
```

Le code affichera à l'écran : *Bonjour \$nom*

Dans le premier exemple, en utilisant "...", c'est la valeur de la variable qui est transmise tandis que dans le second exemple, en utilisant '...', c'est le nom de la variable qui est transmis. En effet, lorsque vous utilisez les apostrophes au lieu des guillemets, la variable n'est pas interprétée comme variable mais comme chaîne de caractères.

Attention :

Lorsque vous utilisez les apostrophes ou les guillemets, si dans votre propre texte vous utilisez les mêmes symboles, alors vous devez les faire précéder d'un antislash. Par exemple : `echo 'vous n\'êtes pas inscrit';`

Exemple 40 Exemple avec erreur

```
1 <?php
2     //-----
3     // Ce code affichera une erreur
4     //-----
5     echo "<a href='http://www.sig.hec.ulg.ac.be' > Site SIG</a>";
6 ?>
```

Exemple 41 Exemple correct

```
1 <?php
2     // -----
3     // Ce code est correct
4     // -----
5     echo "<a href=\"http://www.sig.hec.ulg.ac.be\">Site SIG</a>";
6 ?>
```

5.6 Opérateurs de base

Les instructions les plus élémentaires utilisent les opérateurs mathématiques de base déjà utilisés dans les feuilles d'Excel : + (addition), * (multiplication), - (soustraction), / (division). Ces instructions s'appliquent soit aux variables, soit directement à des valeurs. Le résultat de ces opérations pourra être affecté à des variables à l'aide du symbole d'égalité. Ce symbole sera précédé du nom de la variable qui recevra le résultat et suivi de l'expression à calculer.

Exemple 42 Opérations sur des variables

```
1 <?php
2     $maVar = 3 + ($autreVar * 2) + 1;
3 ?>
```

L'opérateur *modulo*, qui donne le reste de la division entière d'un nombre par un autre, est également très utile. Son symbole est %.

Un autre opérateur qui nous sera bien utile dans ce cours introductif est l'opérateur de *concaténation* de chaînes. Il s'utilise pour accoler des chaînes de caractères (des fragments de texte) les uns aux autres. Cet opérateur est représenté par un point.

Exemple 43 Concaténation

```
1 <?php
2     $maVar = 12;
3     $monTexte = "Résultat = " . $maVar;
4     echo $monTexte;
5 ?>
```

Le code affichera à l'écran : "Résultat = 12", vu que la valeur affectée à \$monTexte vaut "Résultat = 12".

Exemple 44 Concaténation 2

```
1 <?php
2     $nom = "visiteur";
3     echo "Bonjour " . $nom;
4 ?>
```

Le code affichera à l'écran : Bonjour visiteur

Remarque :

Ecrire les variables à l'intérieur des guillemets doubles est équivalent à les écrire à l'extérieur de ceux-ci en les concaténant avec un point.

5.7 Instructions de test IF

Parfois, on désire n'exécuter certaines instructions que si une condition particulière est satisfaite (par exemple, afficher "Bonjour" si l'heure actuelle est avant 18h, "Bonsoir" sinon). L'instruction PHP permettant de tester une condition est le `if`. Il est important d'aligner correctement les instructions au sein d'un code. Ci-dessous, les deux syntaxes acceptées pour l'instruction `if` :

Exemple 45 if avec accolades alignées

```
1 <?php
2     if(condition)
3     {
4         Instruction à exécuter si la condition est réalisée;
5     }
6 ?>
```

Les instructions entre accolades ne seront exécutées que si la condition est réalisée. Sinon elles seront simplement ignorées.

Exemple 46 *if aligné avec l' accolade de fin*

```
1 <?php
2     if(condition){
3         Instruction à exécuter si la condition est réalisée;
4     }
5 ?>
```

Comme vous avez pu le remarquer, deux syntaxes d'alignements sont acceptées. Un exemple de l'utilisation de ces des deux syntaxes a été donné pour l'instruction `if`. Le type d'alignement reste le même pour toutes les autres instructions qui seront vues par la suite. Il est possible d'aligner les accolades les unes par rapport aux autres ou d'aligner le nom de l'instruction principale (dans notre cas le `if`) par rapport à l'accolade de fin.

Il est possible d'enrichir cette syntaxe en ajoutant des instructions alternatives à exécuter si la condition n'est pas réalisée. Dans ce cas, on fera suivre le `if` d'une instruction `else`. De nouveau, deux syntaxes sont possibles.

Exemple 47 *if-else avec accolades alignées*

```
1 <?php
2     if(condition)
3     {
4         Instruction à exécuter si la condition est réalisée;
5     }
6     else
7     {
8         Instruction à exécuter sinon;
9     }
10 ?>
```

Dans ce cas, si la condition est vraie, le premier bloc d'instructions est exécuté, mais pas le second. Par contre, si la condition n'est pas vérifiée, c'est le second bloc qui sera exécuté et non pas le premier.

Exemple 48 *Syntaxe alternative*

```
1 <?php
2     if(condition){
3         Instruction à exécuter si la condition est réalisée;
4     }
5     else{
6         Instruction à exécuter sinon;
7     }
8 ?>
```

Il est possible d'enrichir encore cette syntaxe pour combiner plusieurs tests :

Exemple 49 *Combinaison de plusieurs conditions*


```

1 <?php
2     if(condition1) {
3         Instruction1;
4     }
5     elseif(condition2) {
6         Instruction2;
7     }
8     else{
9         Instruction3;
10    }
11 ?>

```

De nouveau l'autre syntaxe peut également être employée.

Dans cette dernière syntaxe, on pourra placer successivement autant de `elseif` que l'on souhaite et terminer avec ou sans `else`.

5.7.1 Opérateurs de comparaison

Très souvent, une condition consiste à comparer deux valeurs (nombres ou chaînes de caractère). Imaginons que nous voulions afficher "*Bonjour Madame*" si la variable `$sexe` contient la valeur "F" et "*Bonjour Monsieur*" sinon. Il faut donc écrire une condition qui permet de comparer le contenu de la variable `$sexe` avec la chaîne de caractère "F".

Les principaux opérateurs de comparaison utilisés dans les tests sont repris dans le tableau ci-dessous :

Code	Résultat
<code>\$x == \$y</code>	retourne true si \$x est égal à \$y.
<code>\$x != \$y</code>	retourne true si \$x n'est pas égal à \$y.
<code>\$x > \$y</code>	retourne true si \$x est plus grand que \$y.
<code>\$x < \$y</code>	retourne true si \$x est plus petit que \$y.
<code>\$x >= \$y</code>	retourne true si \$x est plus grand ou égal à \$y.
<code>\$x <= \$y</code>	retourne true si \$x est plus petit ou égal à \$y.

Attention :

L'opérateur de comparaison est deux fois le symbole d'égalité et non pas une fois le symbole d'égalité. Le symbole (simple) d'égalité doit uniquement être utilisé pour l'affectation d'une valeur à une variable. Le piège est que l'utilisation du simple `=` dans un `if` ne provoquera pas de message d'erreur mais donnera un résultat souvent surprenant (très souvent égal à vrai).

Exemple 50 Test

```

1 <?php
2 // Si la variable nommée $var est égale à ok
3 if ($var == "ok") {
4     echo "test"; // On affiche le résultat
5 }
6 // Sinon, on affiche un autre message
7 else {
8     echo "refusé";
9 }
10 ?>

```

Exemple 51 *Test 2*

```

1  <?php
2  // Si la variable nommée $var est égale à 1
3  if ($var == 1) {
4      echo "Bravo vous avez trouvé";
5  }
6  // Sinon, si la variable nommée $var est égale à 2
7  else if ($var == 2) {
8      echo "C'est presque ça";
9  }
10 // Sinon, on affiche un autre message
11 else {
12     echo "Ce n'est pas ça veuillez réessayer";
13 }
14 ?>

```

5.7.2 Opérateurs logiques

Parfois, il est utile de tester que plusieurs conditions à la fois sont réalisées, ou encore qu'au moins une parmi plusieurs conditions est réalisée. Dans ce cas, on utilise les *opérateurs logiques* repris dans le tableau suivant.

Code	Résultat
condition1 AND condition2	true si condition1 ET condition2 sont vraies.
condition1 && condition2	idem
condition1 OR condition2	true si condition1 est vraie OU condition2 est vraie.
condition1 condition2	idem
condition1 XOR condition2	true si condition1 est vraie OU condition2 est vraie mais false si les deux sont vraies simultanément.
!condition1	true si condition1 n'est pas vraie

5.8 Les boucles

Si on désire exécuter le même ensemble d'instructions (ou des instructions similaires à une valeur près) jusqu'à ce qu'une certaine condition soit remplie, on utilisera les *boucles*. Différentes syntaxes existent. Les plus courantes utilisent les instructions **while** et **for**.

Détaillons la syntaxe de chacune de ces instructions ainsi que leur usage.

5.8.1 Boucle while

La syntaxe de la boucle **while** est donnée par

```

1  <?php
2  while(condition) {
3      Instruction1;
4  }
5  ?>

```

La boucle **while** exécute les instructions entre accolades **tant que** la condition est vérifiée. Détaillons les opérations effectuées par l'interpréteur lors de la rencontre avec une boucle **while**.

1. Premièrement, la condition est vérifiée. Si cette condition est satisfaite, toutes les instructions entre accolades sont exécutées. Sinon, l'interpréteur passe à la ligne suivant la dernière accolade (après la boucle) et la boucle est alors terminée. Jusque là, le fonctionnement est identique à l'instruction **if**.
2. Néanmoins, avec la boucle **while**, si la condition était vérifiée, lorsque la dernière instruction entre accolades a été exécutée, l'interpréteur ne continue pas à la ligne suivante mais se repositionne juste avant le **while** et teste une nouvelle fois la condition (retourne à l'étape 1) et ainsi de suite jusqu'à ce que la condition ne soit plus vérifiée et que l'interpréteur ne rentre plus dans la boucle.

Attention :

Si la condition est vraie initialement et qu'aucune instruction dans le corps de la boucle ne permet de modifier la valeur de la condition, la procédure ne se terminera jamais (la condition étant toujours vérifiée, le programme exécutera les instructions entre accolades un nombre infini de fois). C'est ce qu'on appelle un *Deadlock*.

L'instruction **while** est la plus générale. Toutes les instructions de boucles pourraient être réécrites en utilisant uniquement un **while**.

5.8.2 Boucle for

La syntaxe de la boucle **for** est donnée par

```
1 <?php
2 for($cpt = $min; $cpt < $max; $cpt++) {
3     Instruction1;
4 }
5 ?>
```

La boucle **for** utilise une variable de type *integer* qui sert de compteur. Elle est ici appelée **\$cpt** (son nom n'a pas d'importance). Détaillons les étapes exécutées par l'interpréteur lors de la rencontre avec une boucle **for**.

1. Le premier élément de la parenthèse de l'instruction **for** est l'*étape d'initialisation*. Lorsqu'il rencontre l'instruction **for**, l'interpréteur exécute cette initialisation. Dans le code ci-dessus, la variable **\$cpt** est initialisée à la valeur **\$min** (qui doit avoir été préalablement définie).
2. Le second élément de l'instruction **for** est la *condition*. L'interpréteur vérifie que cette condition est vérifiée. Ici, la condition à vérifier est que la valeur de la variable **\$cpt** soit inférieure à la valeur **\$max**. Si cette

condition est satisfaite, toutes les instructions entre accolades sont exécutées. Sinon, il passe à la ligne suivant l'accolade finale (après la boucle) et la boucle est terminée.

3. Si la condition est vérifiée, lorsque l'accolade finale est rencontrée, l'interpréteur exécute le troisième élément de l'instruction `for`. C'est l'étape d'incrément. Dans notre cas, l'instruction `$cpt++` signifie que la valeur de `$cpt` est augmentée d'une unité à la fin de chaque passage dans la boucle. L'interpréteur retourne alors à l'étape numéro 2.

Bien que la boucle `while` permette de réaliser toutes les boucles, la boucle `for` permet parfois de réduire la longueur du code.

Remarque :

Notons que l'instruction `$cpt++` est équivalente à l'instruction `$cpt= $cpt+1`. Le symbole `=` correspond à une affectation et non à une égalité, cette instruction est interprétée comme : le contenu de la variable `$cpt` devient le contenu de la variable `$cpt` plus 1. Il est également possible d'utiliser une syntaxe similaire pour décrémenter un nombre ainsi que pour le diviser ou le multiplier. Voici les différentes syntaxes :

Incrément	Décrément	Multiplication	Division
<code>\$cpt = \$cpt + 1</code> <code>\$cpt++</code>	<code>\$cpt= \$cpt-1</code> <code>\$cpt--</code>	<code>\$cpt=\$cpt*2</code> <code>\$cpt *=2</code>	<code>\$cpt=\$cpt/2</code> <code>\$cpt/=2</code>

Exemple 52 Boucles équivalentes avec les `for` et `while`.

```

1 <?php
2 $i = 0; // initialisation
3
4 // condition: la boucle se terminera lorsque la variable $i
  sera égale à 5
5 while($i <= 4) {
6     echo "Boucle numero " . $i . "<br/>";
7     $i++; // Le ++ sert à ajouter 1 à chaque tour de boucle
8 }
9 ?>

```

Remarquons que l'étape d'initialisation est précisée, dans le cas d'une boucle `while`, avant l'instruction. La dernière instruction de l'accolade est l'étape d'incrément. Si cette étape est omise, le programme bouclera infiniment (Dead-Lock).

```

1 <?php
2 // La boucle se terminera lorsque la variable $i sera égale à 5
3 for($i=0; $i<=4; $i++) {
4     echo 'Boucle numero ' . $i . '<br />';
5 }
6 ?>

```

Les deux codes précédents afficheront la même chose à l'écran :
Boucle numero 0

Boucle numero 1

Boucle numero 2

Boucle numero 3

Boucle numero 4

Notons que la variable `$i` est ici utilisée à la fois pour contrôler le nombre de passages dans la boucle, mais également le chiffre affiché à chaque passage.

5.9 Les tableaux - vecteurs

Il est possible de stocker plusieurs valeurs d'un même type derrière un seul nom de variable. La variable est alors un tableau (ou vecteur) composé de plusieurs cases mémoire, chacune numérotée, pouvant stocker une valeur. Quelle est l'utilité d'un tel tableau ? Imaginons que nous voulions stocker en mémoire une liste de produits, par exemple une liste de marques de voitures. Nous pourrions les stocker chacune dans une variable.

```
1 <?php
2 $car1= "Volvo";
3 $car2= "BMW";
4 $car3= "Volkswagen";
5 ?>
```

Imaginons maintenant que nous ayons 300 marques de voitures et non 3, et que nous voulions écrire un programme pour boucler et passer toutes ces marques en revue afin d'en trouver une spécifique. La solution est alors d'utiliser un tableau pour stocker tous ces éléments sous un seul nom de variable et accéder à chaque élément uniquement à l'aide du numéro de case.

Pour définir un tableau et lui affecter des valeurs simultanément, on utilise la syntaxe suivante :

```
1 $cars = array("Volvo", "BMW", "Volkswagen");
```

Pour accéder à un élément du tableau, on fait suivre le nom du tableau par le numéro d'index entre accolades. Notons que les cases sont numérotées à partir de 0 et pas à partir de 1.

Il est également possible de définir un tableau vide et lui assigner ensuite les valeurs une par une manuellement.

```
1 $cars = array();
2 $cars[0] = "Volvo";
3 $cars[1] = "BMW";
4 $cars[2] = "Volkswagen";
```

Il existe également d'autres syntaxes qui permettent d'associer des noms différents aux cases du tableau, mais nous n'insisterons pas sur cela ici.

Exemple 53 *Exemple de tableau*

```
1 <?php
2     $tableau = array("Anne", "Bob", "Charles");
3     echo $tableau[0];
4     // Le mot Anne est affiché
5     $tableau[2] = "truc";
6     // Le mot truc remplace le mot Charles dans le tableau
7 ?>
```

5.10 Fonctions utiles en PHP

Le PHP est un langage puissant qui propose de nombreuses fonctions prédéfinies intéressantes. Il serait impossible de les présenter toutes et le lecteur est invité à consulter le site <http://www.php.net>. En voici une brève sélection :

5.10.1 Date et temps

- `time()` : nombre de secondes depuis le 1/1/1970
- `date(format)` : date du jour formatée (par exemple un format peut être "Y/m/d" pour l'année/mois/jour ou encore "h:i:s" pour heure :minute :seconde)
- `date(format, $unedate)` : date "unedate" formatée

Pour une liste complètes des fonctions liées aux dates et temps, consultez http://www.w3schools.com/php/php_ref_date.asp.

5.10.2 Precision

- `abs()` : valeur absolue d'un nombre
- `floor()` : valeur entière d'un nombre
- `round($nbre, $precision)` : arrondi un nombre à une certaine précision

5.10.3 Chaînes de caractères

- `ereg($pattern, $text)` : la chaîne de caractères `$pattern` est-elle présente dans le texte `$text` ?
- `strlen()` : donne la longueur d'une chaîne de caractères
- `str_word_count()` : compte le nombre de mots dans une chaîne de caractères
- `strtolower()` : convertit en minuscules
- `strtoupper()` : convertit en majuscules

5.10.4 Aléatoire

- `rand()` : génère un nombre entier aléatoire positif
- `rand($min, $max)` : génère un nombre entier aléatoire compris entre `$min` et `$max`.

5.10.5 Tableaux

- `asort()` : trie les éléments d'un tableau par ordre croissant
- `isort()` : trie les éléments d'un tableau par ordre décroissant
- `count()` : donne la taille du tableau `$tableau`.

Exemple 54 Exemples de fonctions utiles

```
1 <?php
2 $mdate = time();
3 $mdateetxt = date("d/m/y" , $mdate);
4
5 $maval = round(17.123 , 2);
6 $maval = round(17.125 , 2);
7
8 $maVariableAleatoire = rand();
9 $maVariableAleatoire = rand(2, 10);
10
11 $tableau = array(1, 3, 5, 7, 9);
12 $maTaille = count($tableau )
13 ?>
```

5.11 Récupération des données d'un formulaire

Tout l'intérêt du PHP est de pouvoir créer des pages à la demande. Pourquoi? Parce que l'utilisateur souhaite souvent des informations spécifiques, parce qu'il veut des réponses à des questions précises, parce qu'il préfère qu'on le considère comme une personne et non pas comme un inconnu. Cela se traduit par des moteurs de recherche dans lesquels les Internautes spécifient leur demande et reçoivent en retour une réponse sur mesure. Cela se traduit aussi en gestion et en marketing par la notion de "*Gestion de la Relation Client*" (CRM en anglais) où le vendeur tente d'identifier au plus tôt l'Internaute pour lui proposer des produits et services sur mesure de sorte à le satisfaire et le fidéliser.

Pour créer de l'interaction avec l'Internaute et lui permettre d'introduire des informations dans une page, nous avons appris à créer des formulaires dans le langage HTML. Nous n'avons pas encore étudié comment récupérer nous-mêmes les données du formulaire pour les traiter. Cela n'est en effet possible qu'avec un langage tel que le PHP. Dans cette section, nous allons apprendre à créer nos propres programmes permettant de récupérer et traiter les données encodées par l'internaute. Nous allons en fait écrire nos propres **action** de formulaires.

Comme vu précédemment, un formulaire est composé de différents objets `<input>`, chacun portant un nom et un type. Chaque objet est l'équivalent d'une variable. Le nom de la variable est défini par **name** et son type par **type**. Par exemple :

```
1 <html>
2 <body>
```

```
3 <form action="welcome.php" method="post">
4   Nom: <input type="text" name="nom"><br>
5   E-mail: <input type="text" name="email"><br>
6   <input type="submit">
7 </form>
8 </body>
9 </html>
```

Lorsque l'internaute remplit ce formulaire et clique sur le bouton `submit`, les données sont envoyées à un fichier PHP intitulé `welcome.php`, qui est spécifié comme action du formulaire. Dans ce fichier, la syntaxe pour récupérer les valeurs rentrées dans le formulaire et les stocker dans des variables est la suivante :

```
1 <?php
2   $valname = $_POST["nom"]; //la méthode était POST
3   $valmail = $_POST["email"];
4   ?>
```

Si la méthode du formulaire était `GET`, il suffit de remplacer `POST` par `GET`.

Lorsque l'internaute clique sur le bouton `submit`, le programme spécifié dans l'action du formulaire est exécuté. Dans cet exemple, la valeur entrée par l'internaute dans l'input ayant pour `name` `nom` est récupérée et stockée dans la variable `valname`. La valeur entrée dans l'input ayant pour `name` `email` est stockée dans la variable `$valmail`. Ensuite, il est possible de manipuler ces variables comme n'importe quelle autre variable PHP (effectuer des calculs sur des données chiffrées entrées par l'utilisateur, afficher les données entrées, effectuer des tests,...).

Une fonction utile dans ce contexte est la fonction `isset()` qui permet de tester si la variable entre parenthèses existe et est non vide. Cette fonction retourne `true` si la variable existe et `false` sinon. Cela permet notamment de tester si l'utilisateur a bien rempli tous les champs avant d'effectuer des opérations sur les données récupérées.

Nous sommes à présent en mesure de créer des formulaires un peu plus complexes : les formulaires *sticky*. Un *formulaire sticky* est un formulaire qui retient les valeurs entrées par l'internaute après qu'il les ait soumis et garde celles-ci dans les cases. Par exemple, si l'utilisateur soumet un formulaire sans préciser tous les champs requis, cela lui évite de devoir compléter à nouveau tous les champs (voir séances d'exercices).

5.12 Données persistantes : cookies et sessions

Il est souvent nécessaire de mémoriser des informations sur l'Internaute tant qu'il est en train de visiter le site et même d'une visite à l'autre. L'exemple le plus parlant est probablement un site de commerce en ligne où l'utilisateur remplit son caddie d'une page à l'autre avant de passer à la caisse puis enfin quitter le site. Conserver une trace de l'identité du visiteur et du contenu de son caddie d'une page à l'autre serait complexe via des formulaires "sticky" et lourd

via des écritures dans une base de données coté serveur. La solution réside dans les cookies et sessions.

5.12.1 Ecriture d'un cookie

Un cookie est un petit fichier texte que le serveur écrit dans l'ordinateur de l'utilisateur. Le langage PHP permet d'écrire et de récupérer des cookies.

Un cookie est créé à l'aide de la fonction `setcookie()`. Pour stocker dans un cookie pendant `$second` secondes, sur l'ordinateur de l'internaute, une valeur `$valeur` dans une "variable" nommée par exemple `varcook`, on écrira :

```
1 <?php
2     setcookie("varcook" , $valeur , time() + $second , "/");
3     $_COOKIE["varcook"] = $valeur ;
4 ?>
```

Attention :

Cette instruction PHP a pour effet de créer automatiquement les marques HTML `<head>...</head>`. Etant donné qu'un seul header est accepté par document HTML, il ne faut pas écrire soi-même de marque `<head>` dans ce cas. De plus, ces instructions doivent impérativement commencer le fichier PHP. Il ne peut même pas y avoir un espace avant le `<?php`.

Pour modifier un cookie, on utilise la fonction `setcookie()` de la même façon (on réécrit le cookie). Pour supprimer un cookie, on utilise la fonction `setcookie()` avec une date d'expiration dans le passé.

5.12.2 Lecture d'un cookie

La lecture d'un cookie se réalise en consultant un "tableau" prédéfini en PHP et dont chaque élément a été initialisé avec les noms des variables stockées dans le cookie. La syntaxe est la suivante

```
1 <?php
2     if (isset($_COOKIE["varcook"] )) {
3         echo $_COOKIE["varcook"] ;
4     }
5 ?>
```

Exemple 55 Exemple de création et lecture de cookie

```
1 <?php
2     $cookie_name = "user";
3     $cookie_value = "John Doe";
4     setcookie($cookie_name , $cookie_value , time()+(86400 * 30) ,
5         "/");
6     $_COOKIE[$cookie_name] = $cookie_value ;
7 ?>
8 <html>
9 <body>
10 <?php
11     if(!isset($_COOKIE[$cookie_name] )) {
```

```

11     echo "Le cookie nommé '". $cookie_name. "' n'est pas défini!";
12     ";
13 }else {
14     echo "Le cookie '". $cookie_name. "' est défini!<br>";
15     echo "Sa valeur est: " . $_COOKIE[$cookie_name];
16 }
17 ?>
18 </body>
19 </html>

```

5.12.3 Création de sessions

On se sert des sessions pour stocker des informations dans des variables afin de pouvoir les utiliser à travers plusieurs pages distinctes. Contrairement à un cookie, les informations ne sont pas stockées sur l'ordinateur de l'utilisateur. Par défaut, les variables de session sont des informations retenues d'une page à l'autre, qui sont détruites lors de la fermeture du navigateur.

Pour créer des sessions, plusieurs fonctions PHP sont utilisées :

- `session_start()` : code à spécifier au début de toutes les pages où on travaille avec une session (stockage de variables de session ou lecture). Ce code doit être placé obligatoirement au tout début de la page, avant la balise `<html>`.
- `session_is_registered("mavar")` : permet de tester si la variable `mavar` a été déclarée comme variable de session. Retourne `true` si `mavar` a été déclarée comme variable de session, `false` sinon.
- `session_id()` : indique le numéro d'identification (unique) de la session
- Pour définir une variable de session ayant pour nom `mavar` et lui affecter la valeur `$value`, la syntaxe est `$_SESSION["mavar"] = $value`.
- Pour détruire la session, le code est `session_destroy()`.

Par exemple, créons la page suivante `demo_session1.php`

```

1  <?php
2  /*Indiquer qu'on travaille avec une session*/
3  session_start();
4  ?>
5  <!DOCTYPE html>
6  <html>
7  <body>
8      <?php
9      /* Définition des variables de session */
10     $_SESSION["favcolor"] = "gris métallisé";
11     $_SESSION["opt"] = "volant cuir";
12     echo "Les variables de session sont définies.";
13     ?>
14 </body>
15 </html>

```

Créons maintenant également une seconde page `demo_session2.php`.

```

1  <?php

```

```

2     session_start();
3     ?>
4     <!DOCTYPE html>
5     <html>
6     <body>
7         <?php
8             /* Afficher les variables de session définies sur l'autre
               page */
9             echo "La couleur désirée est ". $_SESSION["favcolor"] .".<br>
               ";
10            echo "L'option choisie est ". $_SESSION["opt"] .". ";
11            ?>
12        </body>
13    </html>

```

Les variables de session permettent d'améliorer fortement l'interaction avec le client. Entre autres, elles permettent de gérer un panier d'achats et mémoriser les articles souhaités de page en page au sein d'une même session, elles permettent également de mémoriser la langue choisie par l'internaute, le moyen de paiement,... Elles permettent aussi de créer un compteur de visites au sein d'une même session (pour créer un compteur de visites valide d'une session à l'autre même si le navigateur a été fermé entre temps, on utilisera plutôt la notion de cookies détaillée auparavant).

Exemple 56 *Compteur de visites et durée de la visite*

```

1     <?php
2     session_start(); /* On indique que l'on travaille avec une
               session */
3     if(!session_is_registered("count")) {
4         $_SESSION["count"] = 0;
5         $_SESSION["start"] = time();
6         /* Si la variable count n'est pas encore définie,
               on l'initialise à 0 et on met la variable
               start au temps actuel */
7     }
8     else {
9         $_SESSION["count"]++;
10        /* sinon, on augmente le nombre de
               visites count d'une unité */
11    }
12    $sessionId = session_id();
13    ?>
14    <html>
15    <head>
16        <title> Session State Test</title>
17    </head>
18    <body>
19        Le numéro d'identification de cette session
20        est le numéro ( <?php echo $sessionId ; ?> )
21        <br/> compteur = <?php echo $_SESSION["count"] ; ?> .
22        <br/> start = <?php echo $_SESSION["start"] ; ?> .
23
24        <br/>La session a duré
25        <?php
26            $duration = time() - $_SESSION["start"] ;

```

```
30     echo "$duration";  
31     ?> secondes.  
32 </body>  
33 </html>
```

5.13 E-commerce

Vous disposez à présent de tous les outils nécessaires à la création d'un site d'entreprise intégrant la plupart des systèmes de Gestion de Relation Client utilisés par les sites web usuels. En effet, la matière vue jusqu'ici suffit pour créer un site d'e-commerce performant : création d'un site esthétiquement attractif, mise en place d'un système de panier d'achat permettant à l'internaute d'effectuer son shopping en ligne, fidélisation du client en proposant des offres personnalisées, récolte d'informations sur le client via les formulaires et les mécanismes de cookies,...

Il reste cependant une fonction supplémentaire à intégrer pour que ce site d'e-commerce soit complet : permettre à l'internaute de régler directement le montant de ses achats en ligne. Dans cette section, nous présentons la démarche permettant d'intégrer un bouton Paypal (Figure 10) à votre site et permettre des transferts d'argent entre comptes Paypal. Notons que la démarche présentée



FIGURE 10 – Bouton Paypal

ici est basée sur le service offert par Paypal à la date du 25 janvier 2022 et que par conséquent, celle-ci peut avoir évolué depuis.

Attention :

Dans le cadre de ce cours, nous allons créer un compte Paypal fictif (version *sandbox*) pour tester les services offerts par Paypal. Il n'y a pas de transfert réels d'argent ! Dans le contexte de ce cours, veuillez ne travailler qu'avec la version *sandbox* : le logo *sandbox* doit être visible au dessus de toutes les pages Paypal et chaque URL doit commencer par `www.sandbox.paypal...` **Veillez à ne jamais donner votre numéro de carte de crédit, ni toute autre information importante !** Nous déclinons toute responsabilité si vous ne suivez pas ces instructions.

5.13.1 Création de compte

1. Sur le site <https://paypal.com/be>, créez un compte avec votre adresse e-mail usuelle et vérifiez vos mails pour activer le compte (Figure 11).
2. Connectez-vous ensuite sur le site <https://developer.paypal.com> avec le compte précédemment créé. Passez le curseur sur votre prénom en haut

Ouvrez un compte PayPal gratuitement.

Choisissez votre type de compte :

☒ **Compte Particulier**

Achetez et payez en ligne ou envoyez et recevez de l'argent. Sans jamais dévoiler vos coordonnées bancaires.

☐ **Compte Professionnel**

Vendez en ligne, acceptez des paiements et envoyez des factures à vos clients. C'est simple et sécurisé, à tout moment et où que vous soyez.

Continuer

FIGURE 11 – Création de compte Paypal

à droite et cliquez sur l'onglet *Dashboard*. Dans le menu latéral gauche, sous *Sandbox*, cliquez sur *Accounts*.

3. Créez successivement deux comptes fictifs (Figure 12, numéro 1) : un pour le vendeur (Business account) et un pour l'acheteur (Personnal account). Faites bien attention à créer des comptes *custom*. Les informations que vous fournissez pour ces comptes sont fictives, il s'agit de comptes *sandbox*. Vous pouvez donc entrer ce que vous voulez, même une adresse e-mail inexistante. Néanmoins, assurez-vous de retenir le mot de passe et le nom d'utilisateur choisis. Le pays que vous choisissez aura un impact sur les étapes suivantes : **choisissez United States**. Confirmez la sélection d'une carte de crédit. Vous pouvez demander qu'un certain montant apparaisse initialement sur le compte.
4. Les deux comptes créés devraient apparaître dans la liste de votre environnement sandbox. Si vous oubliez le mot de passe, vous pouvez cliquer sur les comptes pour avoir accès aux profils et les éditer.

5.13.2 Création d'un bouton de paiement

1. Sur le site <https://www.sandbox.paypal.com/>, connectez vous avec l'adresse e-mail fictive du compte créé pour le **vendeur** et le mot de passe correspondant (Pas votre compte Paypal avec votre véritable adresse mail).
2. En haut de la page d'accueil, choisissez l'onglet *Pay & Get Paid* puis, en dessous de *Accept Payments*, sélectionnez *PayPal Buttons* (voir figure 13).
3. Parmi les différents types de bouton disponibles, sélectionnez le bouton *Buy Now* (voir figure 14).
4. Remplissez les informations pour les trois étapes :

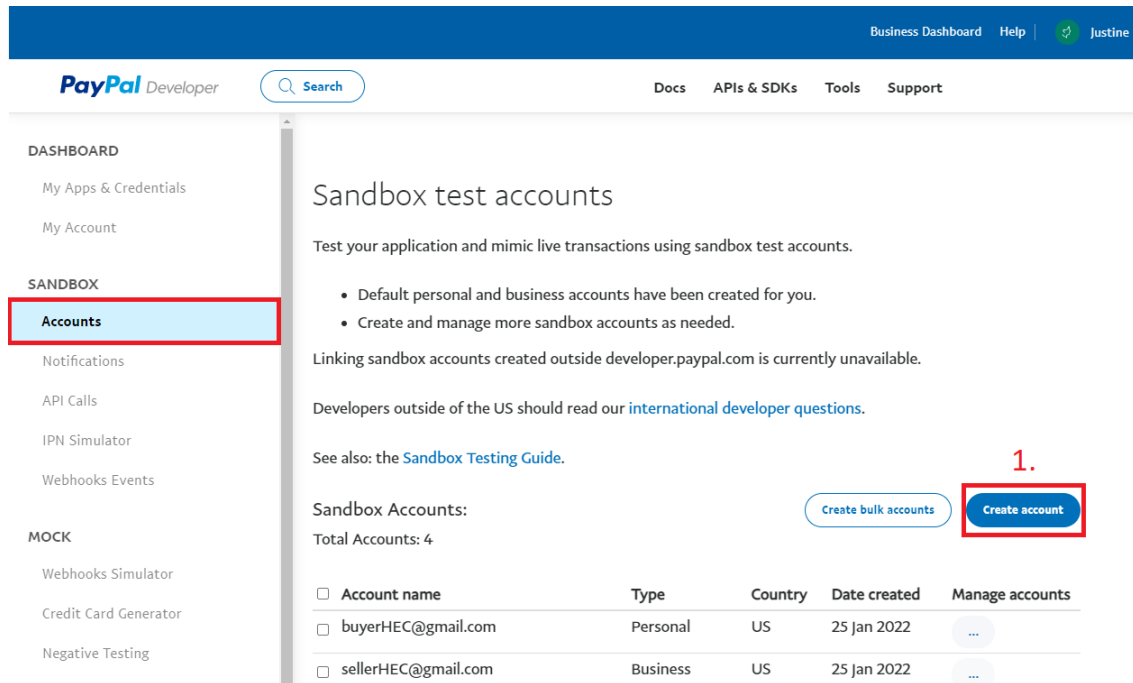


FIGURE 12 – Création de compte Paypal Sandbox

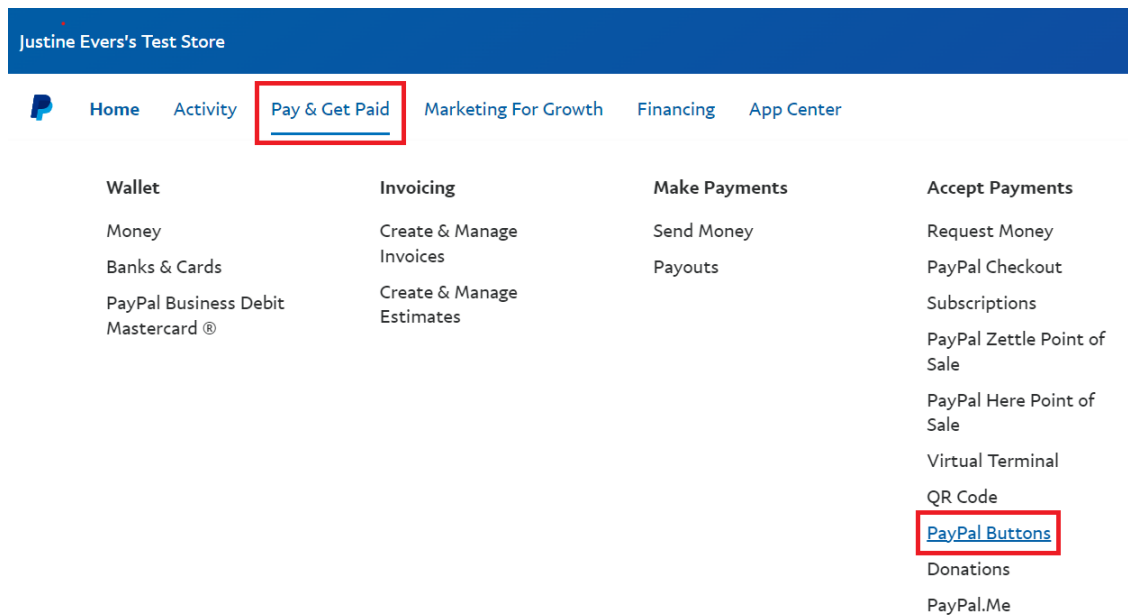
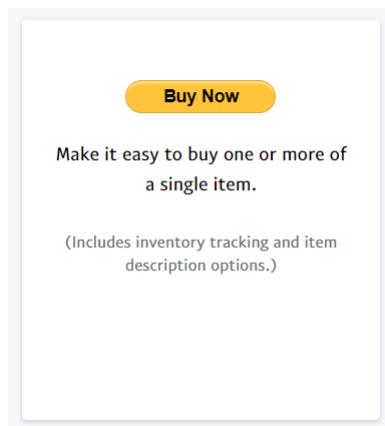
- Etape 1 : important. Par sécurité, n'essayez pas de vendre un produit trop cher. Vous pouvez tester plusieurs options mais restez simple dans un premier temps.
 - Etape 2 : Rien à modifier
 - Etape 3 : Vous pouvez définir ici une adresse de redirection si le paiement est annulé par le client ou quand le paiement est terminé.
5. Une fois que vous avez créé le bouton Paypal, vous allez être redirigé vers une page contenant le code HTML du bouton (voir figure 15). Copiez le code HTML.
 6. Dans votre propre page HTML, collez ce code à l'endroit où vous voulez insérer un bouton Paypal.

Remarque :

Remarquez que le code HTML ainsi obtenu est en fait un formulaire dont l'action est une page du site www.paypal.com. Les différents inputs de type hidden correspondent aux options choisies lors de la création du bouton.

5.13.3 Acheter

Ouvrez la page que vous avez créée et cliquez sur le bouton Paypal. Vérifiez que vous êtes redirigés vers une page SANDBOX. Connectez-vous cette fois avec

FIGURE 13 – Onglet *PayPal Buttons*FIGURE 14 – Bouton *Buy Now*

le compte **client**. Ne donnez aucune information importante ni numéro de carte de crédit à ce stade. ! Confirmez ensuite le paiement fictif.

Add your button code to your webpage

You just created customized HTML code for your button. The final step is to copy the code from this page and paste it into your website editor.

Copy the button code:

1. Click **Select Code**.
2. Right-click and copy the selected code.

If you're working with a website developer, you can paste the button code into an email and send it to your developer now.

Paste the button code in your website editor:

The code must be pasted in the "code" view, where you can view and edit HTML.

1. In your website editor or admin page, open the page where you want to add your button.
2. Look for an option to view or edit HTML.
3. Find the section of the page where you want your button to appear.
4. Right-click and paste your button code into the HTML.
5. Save and publish the page. (The preview function in your editor may not display the button code correctly.)
6. Test the button to make sure it links to a PayPal payment page.

Need more help? [Click here](#) for additional information.

Website

Email

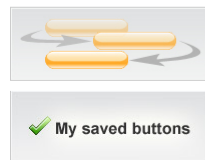
```
<form action="https://www.sandbox.paypal.com/cgi-bin/webscr"
method="post" target="_top">
<input type="hidden" name="cmd" value="_s-xclick">
<input type="hidden" name="hosted_button_id"
value="B29BK4QBZBXY">
<input type="image"
src="https://www.sandbox.paypal.com/en_US/i/btn/btn_buynowCC_LG.g
if" border="0" name="submit" alt="PayPal - The safer, easier way to pay
..."/>

```

Buyer's View

Select Code

Go back to edit this button

Create more buttons

[Create similar button](#) | [Create a new button](#)

Similar: Use the button you just created as a template for another button.
New: Create a completely new button from a blank form.

[Go to My saved buttons](#)

- Edit most characteristics of a button.
- Create a new button that is similar to an existing one.

FIGURE 15 – Code HTML du bouton Paypal

5.13.4 Gestion des comptes

Retournez sur <http://sandbox.paypal.com> et connectez-vous avec votre compte **vendeur** cette fois. Vous pouvez y voir l'historique de vos dernières transactions. Vous devriez avoir reçu le paiement de votre client. A partir de cette page, il est possible de vérifier les transactions, de voir le montant retenu par Paypal, de rembourser des clients,... Vous pouvez aussi faire de même avec le compte client.

6 Bases de données et PHP

Le PHP permet de construire dynamiquement des pages qui correspondent mieux aux besoins des internautes. Ces pages ne deviennent cependant vraiment intéressantes que si elles peuvent manipuler non seulement l'information directement fournie par l'internaute à travers un formulaire, mais également toute information utile connue ou accessible par le gestionnaire du site. Par exemple, Google n'aurait aucun intérêt si pour un mot clef fourni par l'utilisateur, le serveur PHP n'était pas apte à trouver de par lui-même l'ensemble des liens vers des pages associées à ce mot clef.

Dans cette section, nous introduisons d'abord le système de gestion de base de données MySQL et l'interface PHPMyAdmin. Nous présentons ensuite le langage SQL, utilisé par tous les systèmes de gestion de bases de données pour manipuler celles-ci. Enfin, dans la dernière sous-section, nous expliquons comment créer des pages PHP interactives en accédant à des bases de données directement au sein de ces pages.

6.1 MySQL et PHPMyadmin

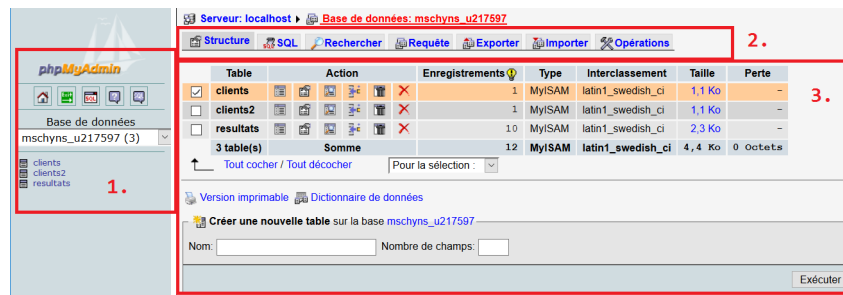
MySQL est un système de gestion de bases de donnée qui fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public que par des professionnels. Tout comme Linux et PHP, c'est un système open. Il peut être téléchargé gratuitement d'Internet et utilisé par tous, aussi bien sous Windows que sous Unix ; ce qui explique d'ailleurs aussi pourquoi tant de fournisseurs d'espaces d'hébergement ont fait ce choix.

MySQL, en tant que logiciel libre, est souvent associé à d'autres outils pour constituer ce que l'on appelle une plateforme LAMP. C'est la combinaison d'un serveur avec système d'exploitation (gratuit) Linux, serveur web (gratuit) Apache, base de données (gratuite) MySQL et langage de programmation (gratuit) PHP. En tant qu'étudiant d'HEC, vous avez accès à un tel serveur http://votre_matricule.php.hec.ulg.ac.be. Une variante du LAMP est le WAMP ; dans laquelle Linux est remplacé par Windows.

MySQL est un SGBD sans interface graphique, la création de tables, de requêtes, se fait en tapant des instructions. Heureusement, à côté de MySQL, on trouve facilement des systèmes de gestion qui savent interagir avec MySQL et facilitent son utilisation. L'interface la plus connue, PhpMyAdmin, est écrite dans le langage de programmation PHP et est accessible via un serveur web. A partir de celui-ci, il est aisé de créer des tables et de réaliser des requêtes simples dessus. Vous accédez à votre base de données HEC, via l'URL <http://php.hec.ulg.ac.be/phpmyadmin>. Attention que pour des questions de sécurité, votre mot de passe pour y accéder n'est pas le même que celui ULg (voir cours).

PhpMyAdmin est composé de trois zones :

1. Dans le bandeau de gauche, on peut choisir sa base de données (si plusieurs existent) et sélectionner une des tables qui la compose. Dans cet



exemple, la base de données sélectionnée est `mschyns_u217597` et trois tables existent pour le moment.

2. Dans la partie supérieure droite, on trouve les commandes applicables à l'objet affiché juste en dessous, généralement la BD ou une table. On y trouve également un chemin cliquable indiquant exactement ce qui est actuellement affiché (dans la figure : base de données : `mschyns_u217597`).
3. Enfin, dans la partie principale à droite s'affiche les informations demandées : contenu de la base de données, structure des tables, contenu des tables, résultats des requêtes ...

Dans la suite, nous expliquerons comment utiliser PhpMyAdmin pour gérer une base de données MySQL. L'objectif n'est pas d'être complet car de nombreuses opérations sont réalisables, dont beaucoup qui ne nous intéressent pas dans un cours introductif et il existe souvent plusieurs chemins pour obtenir un même résultat. Nous nous limiterons à quelques exemples utiles.

6.1.1 Création de tables

Après avoir si nécessaire sélectionné sa base de données, un formulaire s'affiche dans la partie inférieure droite. Il permet de spécifier le nom de la table à créer et le nombre de champs qu'elle contiendra (modifiable par après si nécessaire). Après avoir appuyer sur le bouton *Exécuter*, une nouvelle fenêtre apparaît dans laquelle il est possible de fixer chacun des champs. Chacune des lignes correspond à un champ. Il faut au minimum spécifier un nom de champ (un conseil : évitez les espaces et caractères accentués) et un type de champ.

De nombreux types sont disponibles, les plus courants sont

- INT : nombre entier
- VARCHAR : chaîne de caractères. Dans ce cas il faut également spécifier dans la troisième colonne le nombre de caractères maximum autorisés.
- DATE : date
- BOOLEAN : booléen
- DECIMAL : nombre réel
- TEXT : texte plus long

Le formulaire permet également de spécifier pour chaque champs quelques propriétés supplémentaires : une valeur par défaut (colonne *Défaut*), l'auto-

incrément (colonne *Extra*) pour les nombres entiers et définition des clefs (colonne *clef*).

Si la table et tous les champs nécessaires ont été définis, il suffit d'appuyer sur le bouton *Sauvegarder* afin de créer la table. Par contre, si on a oublié certains champs, il est encore possible d'en ajouter via le bouton *Exécuter* dans le bas du formulaire.

Après avoir créé une table, son nom apparaît dans le bandeau de gauche. Il est alors encore possible à tout moment de modifier sa structure en cliquant sur son nom et en s'assurant que l'onglet *Structure* est bien sélectionné dans la partie supérieure. Lorsque cela est fait, la liste des champs apparaît. Au bout de chaque ligne, on retrouve les mêmes icônes cliquables : afficher le contenu de la table en se limitant à ce champ, modifier la nature du champ, supprimer le champ de la table, définir ce champ comme une clef et d'autres fonctions qui dépassent le cadre de ce cours. On retrouve également un mini-formulaire qui permet d'ajouter d'autres champs.

Pour supprimer une table, on utilise le bouton *Supprimer*.

6.1.2 Insertion de données dans une table

Après avoir sélectionné une table, on peut directement y insérer des données en cliquant sur l'onglet *Insérer*. Un formulaire reprenant la liste des champs, avec les valeurs par défaut, apparaît alors et n'a plus qu'à être complété. L'insertion de lignes dans la table peut se faire une à la fois ou deux par deux.

Une autre technique est utilisable lorsque l'on dispose de toutes les données dans un fichier texte de type CSV¹. PhpMyAdmin est capable d'importer ce type de fichier. Cette opération est réalisable via l'onglet *Importer*. Notons que l'opération inverse à l'importation est également possible ; une ou plusieurs tables peuvent être exportées de MySQL vers un fichier via l'onglet *Exporter*. Les formats d'exportation les plus courants sont le SQL et le CSV.

L'onglet *Vider* permet de vider une table entièrement sans la supprimer (seul son contenu est effacé).

6.1.3 Consultation de table

Après avoir sélectionné la table, l'onglet *Afficher* permet d'afficher TOUT le contenu de la table ; toutes les lignes et toutes les colonnes. Il n'est pas possible ici de limiter à certaines lignes ou colonnes. Notons que l'on peut trier les lignes sur base d'une colonne en cliquant sur le nom de la colonne.

1. Une feuille d'un classeur Excel peut être enregistrée sous ce format via le menu *Enregistrer sous*

Si on veut limiter l'affichage à certaines lignes qui satisfont certains critères, on cliquera plutôt sur l'onglet *Rechercher*. Un semblant de "moteur de recherche" sera alors proposé. Les lignes conservées devront satisfaire des critères. Les critères consistent en la comparaison de la valeur d'un champ avec une valeur fournie par l'utilisateur. Les opérateurs de comparaison disponibles peuvent être sélectionnés via un menu déroulant. Pour les chaînes de caractères, l'opérateur de comparaison EXACT est LIKE. Si on souhaite conserver les lignes pour lesquelles un champ VARCHAR contient exactement ou comme partie une valeur précisée par l'utilisateur, l'opérateur est %LIKE%.

Dans les deux cas, il n'est pas possible via ces onglets de limiter l'affichage à certaines colonnes, de créer artificiellement une colonne supplémentaire, ni de réaliser des requêtes sur plusieurs tables à la fois. Ceci ne peut être fait que via l'onglet "SQL" et l'écriture de commandes en SQL.

6.2 SQL

L'approche classique pour manipuler des bases de données est de le faire via un langage (Language) spécifiant des requêtes (Query) structurées (Structured) à exécuter : le SQL. Tous les SGBD comprennent et utilisent ce langage, même si des interfaces graphiques proposent parfois à l'utilisateur des systèmes plus conviviaux. L'utilisateur attentif aura remarqué qu'à chaque opération exécutée via PhpMyAdmin, cette interface affiche, avant la solution, le code SQL qui a été en réalité exécuté. N'importe quelle requête SQL peut être écrite et exécutée via l'onglet *SQL*.

Outre le fait que connaître un minimum de SQL permet de mieux comprendre le fonctionnement d'un SGBD et donc de le contrôler, cela devient indispensable dans de nombreux cas où il n'existe pas d'interface graphique qui s'y substitue. Ce sera le cas lorsque nous essayerons de créer un site web capable d'interroger des bases de données. Dans ces pages, nous serons obligés de spécifier "à la main" le code SQL nécessaire.

La suite de cette section explique les quelques instructions fondamentales du SQL ; ce qui constitue en réalité le plus gros du langage. Pour plus de détails, l'excellent site <http://www.w3schools.com> enseigne pas à pas toutes les ficelles de ce langage (et de beaucoup d'autres).

6.2.1 Instructions SQL

Consultation d'une table. L'instruction de base pour lire le contenu d'une ou plusieurs tables est le **SELECT**. Le mot clef **SELECT** est immédiatement suivi de la liste des champs à afficher. Une ***** signifie que tous les champs doivent être affichés. Après la liste des champs, le mot clef **FROM** permet de spécifier la table à consulter.

```
1 | SELECT * FROM maTable1
2 | SELECT champ1, champ2 FROM maTable2
```

La première ligne permet d'afficher toutes les lignes et toutes les colonnes de la table nommée **maTable**. La deuxième ligne affiche toute les lignes de la table

maTable2 en limitant l’affichage aux colonnes `champ1`, puis `champ2`.

Pour sélectionner uniquement certaines lignes, on peut ensuite ajouter le mot clef `WHERE`. Seules les lignes qui satisfont la condition spécifiée après ce mot clef seront conservées. Pour combiner plusieurs conditions, les mots clefs `AND` et `OU` peuvent être utilisés. On peut encore ajouter une clause `ORDER BY` (suivi soit de `ASC` soit de `DESC`) et une liste de nom de champs pour demander que le résultat soit trié selon les colonnes spécifiées.

```
1 | SELECT * FROM maTable WHERE condition
2 | SELECT * FROM maTable WHERE condition ORDER BY col5
```

Exemple 57 Sélectionner des lignes

```
1 | SELECT * FROM maTable WHERE (Matricule = "s080158") AND (
   | AnneeAcad="2016")
```

Insertion. L’insertion d’une nouvelle ligne dans une table se fait via l’instruction `INSERT`. Deux syntaxes sont courantes :

```
1 | INSERT INTO table_name VALUES ("value1", "value2",...)
2 | INSERT INTO table_name (champ1, champ2) VALUES ("value1", "
   | value2")
```

Dans le premier cas, on suppose que la valeur de tous les champs à insérer dans la table `table_name` est spécifiée dans l’ordre. Dans le second, seulement les valeurs des champs indiqués après le nom de la table doivent être fournies ; les autres prendront leur valeur par défaut. La seconde syntaxe est particulièrement importante si tous les champs ne doivent pas être encodés initialement ou ne doivent pas être gérés par l’utilisateur.

Notez que les valeurs sont encadrées par des guillemets (même si ce n’est pas indispensable pour les valeurs numériques). Cela pose un problème pour les valeurs qui correspondent à du texte incorporant des guillemets. Il faut dès lors s’assurer de faire précéder chaque apostrophe à l’intérieur d’une valeur par le symbole `\`. Un autre problème peut se poser avec les dates. Suivant le SGBD utilisé, les conventions peuvent varier. La plupart du temps, une date devra être spécifiée au format américain : `aaaa-mm-jj` (4 chiffres pour l’année, 2 pour le mois et 2 pour le jour).

Modification d’une ligne existante. Modifier une ligne déjà présente combine les principes de sélection et d’insertion. L’instruction SQL se nomme `UPDATE`. Pour préciser les lignes qui doivent être mises à jour, le mot clef `WHERE` suivi d’une condition peut être employé. Toutes les lignes qui satisfont cette condition verront leurs valeurs modifiées. Attention que si le `WHERE` est oublié, toutes les lignes de la table seront modifiées.

```
1 | UPDATE table_name SET column_name = "new_value" WHERE
   | column_name = "some_value"
```

Suppression de lignes. Pour supprimer toutes les lignes d'une table ou seulement certaines d'entre-elles, on utilise le mot clef **DELETE**.

```
1 | DELETE FROM table_name WHERE column_name = "some_value"
```

Sans le mot clef **WHERE** et la condition, toutes les lignes de la table seront effacées.

6.3 SGBD et PHP

Pour créer un site interactif performant, le recours à un SGBD est naturel. Pensez par exemple aux bases de données clients, ou aux bases de données contenant votre pseudonyme, mot de passe et autres informations utiles lorsque vous vous connectez en ligne. Le PHP contient une série de fonctions prédéfinies qui permettent d'accéder à des bases de données et de manipuler leur contenu. Le nombre de fonctions nécessaires n'est pas très élevé car seule la liaison avec la base de données doit être assurée en PHP. La manipulation proprement dite du contenu sera à nouveau assurée par le langage SQL, propre aux SGBD. Autrement dit, nous allons insérer du code SQL au sein même de pages PHP.

6.4 Fonctions MySQL

Quel que soit le système de gestion de base de données, trois étapes sont toujours à effectuer

1. Se connecter à la base de données
2. Effectuer les opérations désirées
3. Fermer la connexion à la base de données

Connexion et déconnexion. Pour établir une connexion avec un SGBD, il faut connaître sa localisation et être autorisé, via un identifiant et un mot de passe, à l'utiliser. La fonction `mysql_connect($source, $user, $password)` permet de se connecter au SGBD MySQL. Le premier argument de cette fonction précise la localisation de la base de données. La localisation correspond à l'adresse IP du serveur ou à son nom symbolique. Si le SGBD est sur le même ordinateur que le serveur PHP, on peut utiliser un nom particulier : `localhost`. Les second et troisième arguments de cette fonction précisent le nom d'utilisateur et le mot de passe nécessaires pour accéder au SGBD. Si le SGBD est sur le même ordinateur que le serveur PHP, comme ici, on peut utiliser `$_SERVER["DB_USER"]` comme second argument et `$_SERVER["DB_PASS"]` comme troisième argument. Ces variables contiennent en effet votre nom d'utilisateur et votre mot de passe.

Si la connexion est établie avec succès, la valeur de cette fonction est un identifiant de connexion. Cet identifiant de connexion peut être stocké dans une variable qui sera réutilisée pour la déconnexion.

Si la connexion échoue, la valeur de cette fonction est **FALSE**. Dans ce cas, une fonction utile pour comprendre les raisons de cet échec est la fonction

`mysql_error()`. Cette fonction, utilisée sans argument, retourne un texte expliquant les raisons de la dernière erreur.

Pour se déconnecter du SGBD après utilisation, on utilise la fonction `mysql_close(...)`. L'argument de cette fonction est l'identifiant de la connexion à fermer, i.e. la valeur de la fonction de connexion décrite ci-avant.

Exemple 58 *Connexion et déconnexion à la base de données*

```

1  <?php
2  $connexion = mysql_connect("localhost",$_SERVER["DB_USER"],
3  $_SERVER["DB_PASS"] );
4  if (!$connexion){
5      echo "Erreur: ". mysql_error();
6  }else {
7      echo "Connexion réussie";
8      /* Ajouter ici les instructions de manipulation */
9      mysql_close($connexion );
10 }
11 ?>

```

La connexion au SGBD MySQL se fait grâce à la fonction `mysql_connect()`. La valeur de cette fonction est stockée dans la variable `$connexion`. Si la connexion échoue, la ligne 4 permet d'afficher les raisons de cette erreur. Sinon, on affiche le texte "Connexion réussie". Les instructions de manipulation de données et la fermeture de connexion ne sont effectuées que si la connexion a réussi.

Manipulation de la base de données. Lorsque la connexion au SGBD a été établie avec succès, les bases de données peuvent être manipulées. La première étape consiste à préciser avec quelle base de données vous allez travailler. Pour ce faire, on utilise la fonction `mysql_select_db("mysql_dbname",$connexion)`. Le premier argument de cette fonction est le nom de la base de données à sélectionner. Le second argument est l'identifiant de connexion au SGBD. Cette fonction retourne vrai ou faux selon si la base de données voulue a été trouvée ou non et si vous avez le droit de l'utiliser.

Une fois la base de données sélectionnée, toutes les requêtes MySQL vues précédemment peuvent être effectuées directement dans la page PHP en utilisant la fonction `mysql_query(...)`. L'argument de cette fonction est une chaîne de caractères contenant les requêtes SQL à effectuer. De cette façon, des requêtes de sélection, d'effacement, d'écriture ou de mise à jour peuvent être définies directement au sein de la page PHP.

Exemple 59 *Requête SQL en PHP*

```

1  mysql_select_db( "mschyns_" . $_SERVER['DB_USER'] );
2  mysql_query("DELETE * FROM etudiants WHERE id='s080158'") ;

```

Ces deux instructions peuvent être insérées à la ligne 7 de l'exemple 58 pour choisir la base de données portant le nom `mshyns_votrematricule`. Dans cette base de donnée, dans la table `etudiants` tous les champs des lignes satisfaisant

la condition `id='s080158'` seront supprimés. Attention qu'il est important de bien alterner les guillemets et les apostrophes puisque les instructions SQL sont précisées au sein de guillemets.

Pour des requêtes d'effacement, d'écriture ou de mise à jour de la base de données, il n'y a plus rien à faire. Par contre, si la requête génère des résultats, par exemple une instruction `SELECT`, il faut encore les analyser et les utiliser (les afficher, les tester dans une condition,...). La fonction `mysql_query` retourne un résultat sous forme brute qui doit être stocké dans une variable pour être traité par la suite. Ce résultat brut ressemble à un tableau dont chaque ligne représente une ligne de la table et chaque colonne correspond à un champ.

L'idée est alors de récupérer les lignes de cette table et les afficher ou les stocker dans un tableau PHP. La fonction `mysql_fetch_row(...)` prend comme argument le résultat de la requête (la variable `$result` dans l'exemple 58) et en récupère une ligne à la fois. A chaque appel de cette fonction, la ligne suivante non encore lue du tableau est lue. Cette fonction retourne un vecteur PHP dont le premier élément (position 0) contient la valeur du premier champ requis dans la requête SQL `SELECT` et ainsi de suite. Lorsqu'il n'y a plus aucune ligne à lire, la fonction retourne la valeur `FALSE`.

Exemple 60 Requête SELECT en PHP

```
1 // Récupération des données dans les tables
2 mysql_select_db( "mschyns_" . $_SERVER['DB_USER'] );
3 $result = mysql_query( "SELECT id, email FROM etudiants WHERE id
   = 's080158'" );
4 if (!$result) {
5     echo 'Impossible d\'exécuter la requête : ' . mysql_error();
6 } else {
7     $row = mysql_fetch_row($result );
8     echo $row[0];           // 42
9     echo $row[1];           // La valeur du champ email
10 }
```

Ces instructions, insérées à la ligne 7 de l'exemple 58 permettent de récupérer les champs `id` et `email` des lignes pour lesquelles le champ `id` est égal à 42. Si la requête a échoué, la variable `$result` contient la valeur `FALSE` et un message est affiché. Si la requête réussit, la fonction `mysql_fetch_row()` récupère la première ligne satisfaisant cette requête et la stocke dans le vecteur PHP `$row`.

Souvent, le résultat comprendra plusieurs lignes. Comme la fonction `mysql_fetch_row` ne récupère qu'une seule ligne à la fois, les lignes doivent alors être successivement récupérées à l'aide d'une boucle. Pour déterminer quand la boucle doit être stoppée, soit on vérifie que la fonction `mysql_fetch_row` ne retourne pas la valeur `FALSE`, soit on détermine a priori le nombre de lignes dans le résultat. Pour ce faire, la fonction `mysql_num_rows(...)` prend comme argument le résultat de la requête (`$result`) et retourne le nombre de lignes de ce résultat.

Exemple 61 Récupération de plusieurs lignes


```

1 // Récupération des données dans les tables
2 mysql_select_db( "mschyns_" . $_SERVER['DB_USER'] );
3 $result = mysql_query("SELECT id, email FROM etudiants WHERE id
   = '42'");
4 if (!$result) {
5     echo 'Impossible d\'exécuter la requête : ' . mysql_error();
6 } else{
7     for($i=0; $i < mysql_num_rows( $result ); $i++){
8         // Lecture de la première ligne non encore lue
9         $row = mysql_fetch_row($result );
10        echo $row[0] ." " . $row[1]. "<br />";
11    }
12 }

```

Pour terminer avec les fonctions fondamentales MySQL, il existe une variante de la fonction `mysql_fetch_row`, la fonction `mysql_fetch_assoc(...)` qui prend également comme argument le résultat brut de la requête (`$result`) et retourne un vecteur PHP contenant la première ligne non encore lue du résultat. A la différence de `mysql_fetch_row`, on n'accède pas aux éléments du vecteur en indiquant le numéro de position, mais en donnant comme indice le nom du champ.

Exemple 62 Exemple complet (variante)

```

1 $connexion = mysql_connect("localhost", $_SERVER["DB_USER"] , $_
   _SERVER["DB_PASS"] )) ;
2 if (!$connexion ){
3     echo "Erreur: " . mysql_error();
4     exit; // Arrête le programme ici
5 }
6 echo "Connexion réussie";
7
8 mysql_select_db("mydbname");
9 // Vérification de la connexion à la base de donnée
10 if (!mysql_select_db("mydbname")) {
11     echo "Impossible de sélectionner la base mydbname : " .
        mysql_error();
12     exit;
13 }
14
15 // Exécution de la requête
16 $sql = "SELECT user_id, fullname, user_status FROM une_table
        WHERE user_status = 1";
17 $result = mysql_query($sql);
18 if (!$result) {
19     echo "Impossible d'exécuter la requête ($sql) dans la base :
        " . mysql_error();
20     exit;
21 }
22 // Vérification que le résultat de la requête contient des
    lignes
23 if (mysql_num_rows($result) == 0) {
24     echo "Aucune ligne trouvée, rien à afficher.";
25 } else{
26     /* Tant qu'une ligne existe, placer cette ligne dans la
        variable $row et afficher les éléments */

```

```
27 | while ( $row = mysql_fetch_assoc($result ) ) {  
28 |     echo $row["user_id"] ;  
29 |     echo $row["fullname"] ;  
30 |     echo $row["user_status"] ;  
31 | }  
32 |  
33 | // Fermeture de la connexion  
34 | mysql_close($connexion);
```

Troisième partie

Java

7 Introduction

7.1 Les langages de programmation et le Java

Il existe de nombreux langages de programmation : Fortran, Basic, Pascal, Python, C, C++, C#, Java... Le Java (à ne pas confondre avec Javascript) est le plus récent de la liste précédente et parmi ceux les plus cotés aujourd'hui (avec le C++ dont il s'inspire à la base). Tous ces langages permettent de créer des applications (logiciels) pour les ordinateurs ou assimilés (gsm, tablettes...).

Ces langages doivent être "compilés" pour pouvoir être exécutés. La compilation transforme le code source (Java) en un nouveau fichier qui sera exécutable sur un ordinateur. Il y a donc séparation entre la source et l'exécutable. Un programme spécifique, *le compilateur*, est chargé de vérifier la qualité du code et de le transformer en une version exécutable. Le PHP, au contraire, était "interprété" au moment de l'exécution (un seul fichier lisible par "tous") par *l'interpréteur* (le serveur web en général). Cette différence implique que les programmes en Java ne nécessiteront plus un serveur web et seront exécutables directement sur nos ordinateurs.

Le Java se veut très ouvert. Les programmes compilés en Java peuvent être exécutés "directement" sur tous les systèmes d'exploitation (Windows, Mac OS, Linux, Unix...). Il suffit pour cela qu'un environnement Java soit présent sur l'ordinateur où le programme compilé sera exécuté. On trouve gratuitement sur le web ces environnements sous les noms JRE (*Java Running Environment*) ou JDK (*Java Development Kit*). Le JRE est la version de base pour ceux qui veulent essentiellement utiliser des programmes écrits en Java (et en écrire des versions simples). Le JDK est une version plus complète pour ceux qui veulent pouvoir écrire des programmes Java avancés. Le premier rôle du JRE (ou du JDK) est de servir d'interface/interprète entre votre programme compilé Java et votre système d'exploitation spécifique. C'est une des spécificités du Java : les programmes écrits dans ce langage doivent être facilement portables sur plusieurs systèmes d'exploitation. Au contraire, un code C++ compilé sous Windows ne fonctionnera que sous Windows. Il faudra le recompiler sous MacOS pour obtenir une autre version compatible Mac.

Dans cette partie, vous allez vite constater que la plupart des langages se ressemblent énormément. Ce que vous avez appris en PHP pourra rapidement être transposé au Java ou à d'autres langages de programmation.

7.2 L'environnement de programmation et Eclipse

Le point de départ de tous les programmes est le code source. Que cela soit du PHP, du Java, du C++ ou tout autre langage, la première étape est d'utiliser un logiciel de traitement de texte pour écrire le code. Nous pourrions utiliser Ms Word ou Libre Office, mais ces logiciels de traitement de texte ne sont pas les plus adaptés à cette tâche. Il est préférable d'utiliser un éditeur capable de reconnaître le langage utilisé et de nous aider dans son écriture. Par exemple, l'éditeur mettra les mots clefs en couleur et suggérera des parties de code. Pour le PHP, nous utilisons le NotePad++. Pour le Java, nous allons utiliser un environnement plus riche : Eclipse.

A noter que le Notepad++ reconnaît aussi le Java et mettra en évidence les mots clefs. Cependant, contrairement à Eclipse, le NotePad++ se limite à être un éditeur de texte. Eclipse est lui un environnement complet de programmation. Il intègre entre autres aussi les outils de compilation et de debugging (outils pour repérer les erreurs et vérifier le code).

Eclipse a d'autres gros avantages : gratuit (disponible sur le web), compatible et identique pour les différents OS (Windows, Mac, Linux...), réputé (avec une large communauté d'utilisateurs),...

7.3 Installation du JDK et d'Eclipse

Dans cette partie, les grandes étapes de l'installation du JDK et d'Eclipse vont être présentées. Veuillez suivre l'ordre des étapes !

7.3.1 Installation du JDK

Remarque préliminaire : les programmes écrits en Java sont très courants. Il est possible qu'un JRE (ou JDK) soit déjà installé sur votre ordinateur ! Tentez néanmoins l'installation pour avoir la dernière version complète.

La première étape est d'installer le JDK. Pour ce faire, allez à l'adresse URL suivante :

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>
et sélectionnez votre système d'exploitation (Figure 16).

Les différentes versions disponibles du JDK pour votre système d'exploitation apparaissent. Veuillez à télécharger la bonne version du JDK en fonction du système d'exploitation (Linux, Mac, Windows, ...) installé sur votre ordinateur. Par exemple, pour les ordinateurs sous Windows veuillez télécharger la version « x64 Installer » (OS 64 bits récents) du JDK.

Une fois le téléchargement terminé, double-cliquez sur le fichier `.exe` afin de lancer l'installation du JDK. Appuyez toujours sur le bouton Suivant. A un moment donné, l'assistant d'installation vous demande le répertoire par défaut où devra être installé le JDK. Choisissez le répertoire de votre choix (ou laissez

Java SE Development Kit 17.0.2 downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components in the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

Linux macOS Windows

Product/file description	File size	Download
Arm 64 Compressed Archive	171.49 MB	https://download.oracle.com/java/17/latest/jdk-17_linux-aarch64_bin.tar.gz (sha256 🔗)
Arm 64 RPM Package	153.5 MB	https://download.oracle.com/java/17/latest/jdk-17_linux-aarch64_bin.rpm (sha256 🔗)
x64 Compressed Archive	172.69 MB	https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.tar.gz (sha256 🔗)
x64 Debian Package	148.25 MB	https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.deb (sha256 🔗)
x64 RPM Package	155.11 MB	https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.rpm (sha256 🔗)

FIGURE 16 – Sélectionnez votre système d'exploitation

celui qui vous est proposé). Continuez l'installation en acceptant les options par défaut du programme. A la fin de l'installation, appuyez tout simplement sur le bouton Terminer (Finish).

L'installation du JDK est terminée.

7.3.2 Installation d'Eclipse

Remarque préliminaire : Eclipse se base sur Java pour fonctionner. Il est impératif d'installer le JRE ou JDK en premier! (A noter qu'Eclipse est déjà installé dans les salles informatiques et que vous ne devez pas suivre cette procédure à ces endroits)

La deuxième étape est d'installer Eclipse. Eclipse peut être téléchargé depuis cette adresse : <https://www.eclipse.org/downloads/packages/release/2021-12/r/eclipse-ide-java-developers>. Sélectionnez la version adéquate pour votre système d'exploitation (Figure 17)

Une fois le fichier .zip téléchargé, il suffit de le désarchiver dans un de vos dossiers au choix sur votre disque dur. Il n'y a pas de processus d'installation spécifique à exécuter. Une fois décompressé, en double cliquant sur l'exécutable "eclipse" (avec une boule comme icône), l'environnement démarre immédiatement (il vous demandera juste où il peut placer vos fichiers de travail). Cela veut aussi dire que vous pouvez l'installer où vous "voulez" sans avoir besoin d'être administrateur de la machine.



FIGURE 17 – Version d'Eclipse

8 Les bases du langage Java

8.1 Démarrer avec Eclipse

Au démarrage, Eclipse vous demande de spécifier (ou confirmer) un répertoire de travail. C'est dans celui-là que tous vos codes sources et programmes seront enregistrés (donc conservez le même d'une fois à l'autre!). La toute première fois, un bel écran de présentation apparaîtra. Vous pouvez le fermer pour aller directement à l'espace de travail. Le menu de cette fenêtre devrait ressembler à celui-ci (Figure 18)

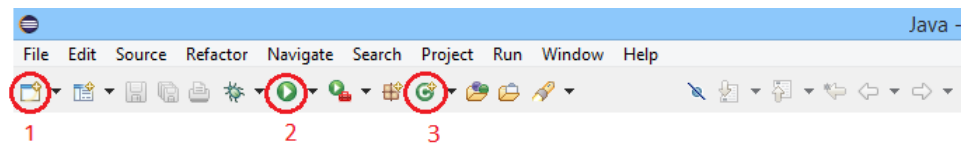


FIGURE 18 – Menu Eclipse

Parmi l'ensemble de ces boutons, 3 vont principalement être utilisés tout au long de ce cours :

1. Le premier bouton permet de créer de nouveau projet Java ainsi que bien d'autres éléments utilisés lors du développement d'un logiciel Java (Package, Class, ...). Cliquer sur ce bouton revient à faire Fichier > Nouveau.
2. Le deuxième bouton permet de compiler et de tester votre code Java.
3. Le troisième bouton est un raccourci utile lors de la création de Classe (Class). Il est également possible de créer une nouvelle classe grâce au premier bouton. Cela revient à faire Fichier > Nouveau > Classe.

Pour créer automatiquement la structure de votre premier projet Java, veuillez cliquer sur le premier bouton et choisir l'icône Java Project (Figure 19).

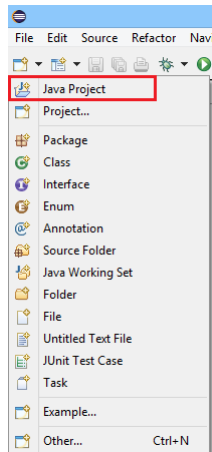


FIGURE 19 – Création de Projet Java - étape 1

Dans la fenêtre suivante (Figure 20), veuillez donner un nom à votre projet, vérifiez que la case sous l'encadré *Module* est bien décochée, puis appuyez sur Terminer (Finish). Notez que tout fichier contenant du code Java doit avoir l'extension '.java' pour pouvoir être exécuté par Eclipse :

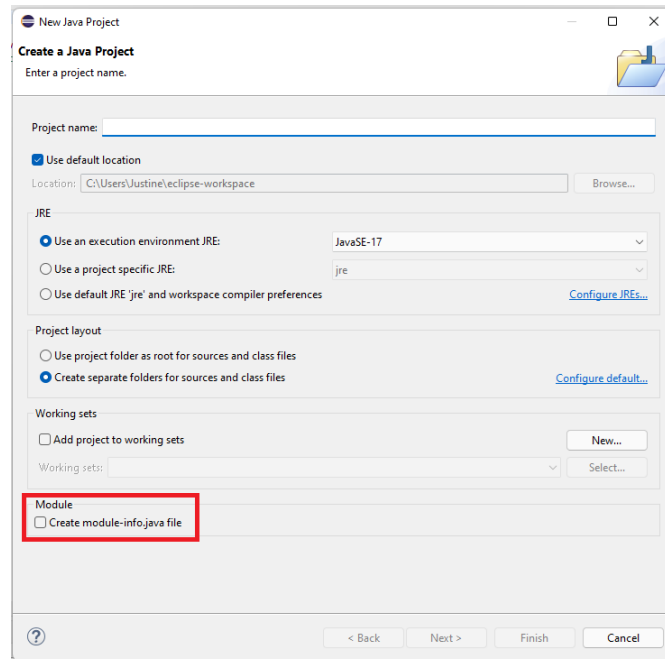


FIGURE 20 – Création de Projet Java - étape 2

Vous pourrez voir apparaître, au niveau du côté gauche de l'écran, votre nouveau projet.

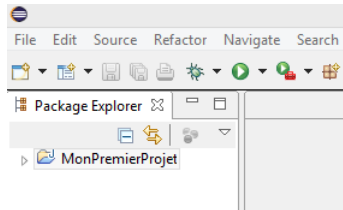


FIGURE 21 – Nouveau Projet

Votre projet Java est créé, mais rien qu'avec ça, vous ne savez pratiquement rien faire. Avant de pouvoir continuer, il va falloir créer un premier fichier qui contiendra par la suite votre code. Le Java est un langage orienté objet. Nous ne rentrerons pas dans les détails maintenant, mais cela implique qu'une organisation très précise des fichiers est attendue. En gros, tout code qui sera utile pour traiter un même concept, une même problématique, un même "objet" sera rassemblé dans ce que l'on appelle une classe. Retenons simplement qu'une classe est un ensemble de codes contenant plusieurs instructions que doit effectuer votre programme. Nous ne travaillerons dans ce cours qu'avec une seule classe à la fois. Il n'y aura ici qu'un seul fichier (portant le même nom que votre classe). Pour créer ce premier fichier, cliquez soit sur le premier bouton et choisissez l'icône Class, ou cliquez tout simplement sur le troisième bouton. Une nouvelle fenêtre apparaît (Figure 22)

Dans cette fenêtre, le premier encadré permet de voir où seront enregistrés les fichiers Java. Dans le second encadré, veuillez donner un nom à votre classe. Le troisième encadré vous demande si votre classe a quelque chose de particulier. Cochez la case `public static void main(String[] args)` (nous reviendrons plus tard sur ce point). Cliquez ensuite sur Terminer (Finish). Votre classe est créée et vous allez enfin pouvoir commencer à programmer !

8.1.1 Environnement Eclipse

La fenêtre principale se lance, comme dans la Figure 23. Dans l'encadré de gauche (en vert), vous trouverez le dossier de votre projet ainsi que son contenu. Ici, vous pourrez gérer votre projet comme bon vous semble (ajout, suppression ...). Dans l'encadré positionné au centre (en bleu), c'est ici que nous écrirons nos codes source. Dans l'encadré du bas (en rouge), c'est là que vous verrez apparaître le résultat de vos programmes, ainsi que les erreurs éventuelles. C'est également là que vous encoderez d'éventuelles informations demandées pouvant être récupérées par votre programme. Et pour finir, c'est dans l'encadré de droite (en orange), dès que nous aurons appris à coder nos propres fonctions et nos objets, que la liste des méthodes et des variables sera affichée.

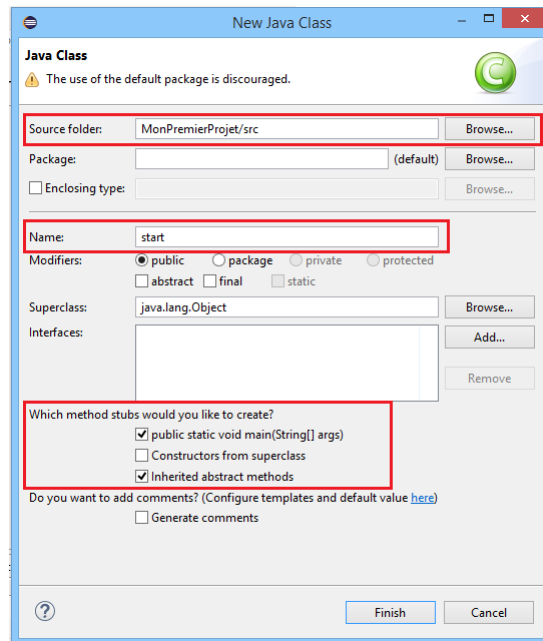


FIGURE 22 – Création d'une classe

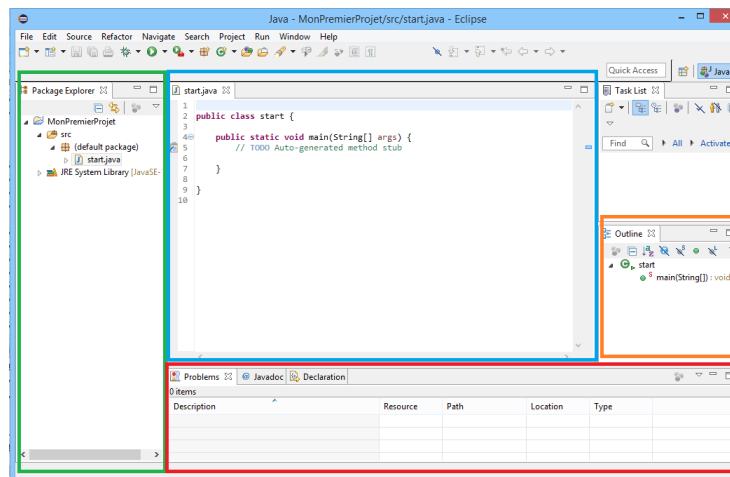


FIGURE 23 – Fenêtre principale Eclipse

Notez que les fichiers .java sont en fait de simples fichiers texte dont l'extension a été changée. Vous pouvez donc les ouvrir, les créer ou encore les mettre à jour avec le Bloc-notes de Windows, par exemple. Cela implique que, si vous le souhaitez, vous pouvez écrire des programmes Java avec le Bloc-notes ou encore

avec Notepad++.

8.2 Structure de base

Un code Java de base correctement écrit ressemble à celui présenté ci-dessous. Chaque instruction à interpréter doit être suivie par un point-virgule. Nous obtenons donc la syntaxe suivante :

```
1 public class Start {  
2     public static void main(String[] args) {  
3         instruction1;  
4         instruction2;  
5         instruction3;  
6     }  
7 }
```

Détaillons les différents éléments présents dans un programme.

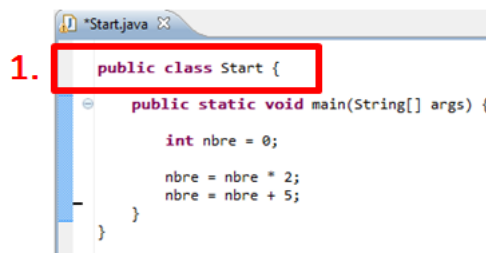


FIGURE 24 – Classe

Le premier élément (Figure 24) est la déclaration de la classe que vous aviez créée précédemment (souvenez-vous que lors de la création de la classe, vous l'aviez appelée Start).

Les classes se retrouvent partout dans Java. Elles vont principalement contenir deux choses :

- des variables (comme en PHP) utilisables partout dans la classe et qui définissent la nature et l'utilité de la classe. Par exemple, si la classe est sensée représenter un véhicule, on trouvera des variables pour identifier la marque, le modèle, la vitesse...
- des méthodes. Dans d'autres langages, on appellerait cela des fonctions ou procédures. Les méthodes contiennent les instructions à exécuter pour atteindre un objectif que l'on se fixe. On pourrait définir pour notre véhicule une méthode "avancer", "tourner"... Vous devez savoir que tous les programmes Java sont composés d'au moins une classe. Elle doit contenir une méthode appelée `main` : c'est celle qui sera exécutée au démarrage du programme. Eclipse l'a créée automatiquement pour nous. (Figure 25) On peut facilement reconnaître les méthodes car celles-ci sont suivies de parenthèses. D'autre part, afin de pouvoir utiliser une méthode dans votre programme, il est nécessaire de la déclarer. On appelle cette déclaration la signature de la méthode :

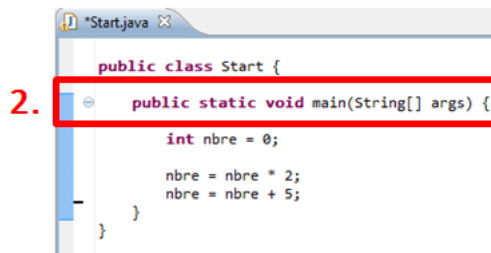


FIGURE 25 – Méthode

```
1 | public static void main(String [] args)
```

Une méthode contient un en-tête (celui-ci va être en quelque sorte la carte d'identité de la méthode), un corps (le contenu de la méthode, délimité par des accolades) et une valeur de retour (le résultat que la méthode va retourner).

Pour comprendre ceci, repensez aux fonctions Excel (hé oui, c'est en réalité très similaire avec de la programmation derrière). Une fonction Excel classique calcule un résultat qui apparaîtra ou sera utilisé dans la cellule du tableau où la formule apparaît. De même, une méthode peut calculer quelque chose qui sera utilisé ailleurs comme résultat de l'exécution de la méthode. Le type de ce "quelque chose" doit être spécifié initialement. Cela peut être un nombre, un texte, autre chose ou rien. Dans l'exemple, `void` indique "rien". Ensuite, comme pour une fonction dans Excel, il peut y avoir des arguments entre les parenthèses. Dans notre exemple, la méthode peut recevoir un tableau de chaînes de caractères `String[] args`. Dans le cadre de ce cours, vous utiliserez principalement cette déclaration de méthode. Mais sachez qu'il en existe d'autres.

Comme pour les classes, les méthodes commencent et se terminent par des accolades. C'est à l'intérieur de celles-ci que vous allez écrire votre code Java.

Le troisième élément de ce code (Figure 26) correspond aux différentes instructions qu'il doit exécuter. Ce code peut servir à calculer un résultat final (ce que retournerait une fonction) ou avoir un effet immédiat (comme une macro dans Excel).

Dans notre exemple, le programme contient une déclaration de variable (`int nbre = 0`) ainsi que 2 instructions. Comme vous pouvez le voir, chaque ligne de ce code se termine par un point-virgule. Ce caractère est obligatoire en Java et son oubli cause une erreur de compilation du code. Veillez donc à ne jamais l'oublier !

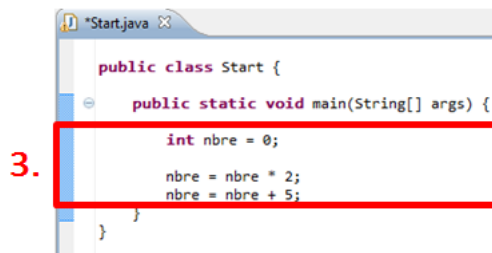


FIGURE 26 – Programme simple

8.2.1 Les commentaires

Il est utile de mettre des commentaires dans votre code afin de le rendre plus compréhensible. La syntaxe est la même qu'en PHP. Les commentaires sur une seule ligne commencent par deux slash //. Tout ce qui se trouve sur cette ligne est considéré comme étant un commentaire. Les commentaires multi-lignes commencent par /* et se terminent par */. Tout ce qui se trouve entre ces symboles est considéré comme étant un commentaire.

Exemple 63 *Exemple de commentaire sur une seule ligne*

```
1 public class start {
2     public static void main(String[] args) {
3         // Voici un commentaire sur une seule ligne
4         // Encore un autre
5     }
6 }
```

Exemple de commentaire multi-lignes

```
1 public class start {
2     public static void main(String[] args) {
3         /* Voici le début d'un commentaire multi-lignes
4            Le commentaire continue sur les lignes suivantes */
5     }
6 }
```

8.3 Les variables

Le Java étant un langage de programmation (tout comme le PHP), des données peuvent être représentées (stockées) dans des variables. En Java, contrairement au PHP, les variables doivent impérativement être déclarées et leur type spécifié avant de pouvoir être utilisées dans des instructions. Les variables peuvent être de plusieurs types.

8.3.1 Variables de type numérique

Le tableau suivant reprend les différents types de variable numérique que l'on rencontre en Java, ce qu'ils peuvent stocker, ainsi qu'un exemple de déclaration.

Type	Contenu	Déclaration
<code>int</code>	entier entre -2^{109} et 2^{109}	1 <code>int nbrArticles;</code> 2 <code>nbrArticles=156;</code>
<code>float</code>	réel	1 <code>float pi;</code> 2 <code>pi=3.141592653;</code>
<code>double</code>	réel (plus de décimales)	1 <code>double nbre;</code> 2 <code>nbre=0.33333333;</code>

8.3.2 Variable de type caractère

Type	Contenu	Déclaration
<code>char</code>	caractère	1 <code>char moncaractere;</code> 2 <code>moncaractere='A';</code>

8.3.3 Variable de type booléen

Le type `boolean` ne peut contenir que deux valeurs : `true` (vrai) ou `false` (faux), sans guillemets (ces valeurs sont natives dans le langage, il les comprend directement et sait les interpréter).

```
1 boolean condition;
2 conditon=true;
```

On peut également déclarer une variable en lui affectant directement une valeur initiale. Pour ce faire, il suffit d'utiliser la syntaxe suivante :

```
1 char moncaractere='A';
2 int entier=1;
```

Et lorsque nous avons plusieurs variables d'un même type, elles peuvent être déclarées en une seule fois de la manière suivante :

```
1 int nbre1 = 2, nbre2 = 3, nbre3 = 0;
```

Note :

Une variable est utilisable dans toutes les lignes qui suivent sa déclaration mais uniquement dans le même "bloc" d'instructions (dans la même classe, dans la même méthode, dans un même bloc dépendant d'un `if`, `for` ou `while`, bref au sein des accolades dans lesquelles elle est déclarée).

8.4 Chaînes de caractères

Il existe aussi le type `String` qui permet de stocker une chaîne de caractère. La différence est qu'il s'agit d'une variable d'un type plus complexe que l'on appelle objet. Vous verrez que celle-ci s'utilise un peu différemment des variables précédentes. Voici plusieurs syntaxes possibles pour la déclaration d'un `String` :

Exemple 64 *Déclaration d'un String*

```
1 //Première méthode de déclaration
2 String phrase;
3 phrase = "J'aime le Java";
4
5 //Deuxième méthode de déclaration
6 String str = new String();
7 str = "Une autre chaîne de caractères";
8
9 //Troisième méthode de déclaration
10 String string = "Une autre chaîne";
11
12 //Quatrième méthode de déclaration
13 String chaine = new String("Et une de plus !");
```

Note : Lors de l'initialisation, on utilise des guillemets doubles.

8.4.1 Conventions de nommage en Java

Vous avez certainement remarqué que, contrairement aux variables, le type `String` commence par une majuscule. En Java, il y a en effet certaines conventions de nommage que l'on tentera de respecter au maximum :

- tous vos noms de classes/objets doivent commencer par une majuscule ;
- tous vos noms de variables doivent commencer par une minuscule ;
- si le nom d'une variable est composé de plusieurs mots, le premier commence par une minuscule, le ou les autres par une majuscule, et ce, sans séparation . Par exemple `unExempleDeVariable` ;
- on évitera d'utiliser les accents.

Notez que le langage Java est sensible aux majuscules et minuscules, aussi bien pour le nom des variables que pour le nom des objets et méthodes employés.

8.5 Afficher à l'écran

Pour pouvoir afficher du texte à l'écran (dans l'encadré rouge de la Figure 23, i.e. la *console*), il est nécessaire d'utiliser la méthode `print()` qui utilise l'objet `out` de la classe `System`. La syntaxe d'affichage d'un texte à l'écran se présente sous la forme suivante :

```
1 | System.out.print("Ce que vous désirez à l'écran");
```

Littéralement, cette ligne de commande signifie "la méthode `print()` va écrire "Ce que vous désirez à l'écran" en utilisant l'objet `out` de la classe `System`".

Détaillons un peu plus ceci.

- `System` : ceci correspond à l'appel d'une classe qui se nomme `System`. C'est une classe qui permet surtout d'utiliser l'entrée et la sortie standard, c'est-à-dire la saisie clavier et l'affichage à l'écran (dans la console).
- `out` : objet de la classe `System` qui gère la **sortie** standard.

- **print** : méthode qui écrit dans la console le texte passé en paramètre (entre les parenthèses).

Exemple 65 *Exemple d’affichage*

```
1 public class start {  
2     public static void main(String[] args) {  
3         System.out.print("Hello World!");  
4     }  
5 }
```

Cette méthode permet également d’afficher des variables qui ont été initialisées au sein du code Java. La syntaxe d’affichage d’une variable est la suivante :

```
1 System.out.print(nom_variable);
```

Exemple 66 *Exemple d’affichage de variable*

```
1 public class start {  
2     public static void main(String[] args) {  
3         int mavariable = 8;  
4         System.out.print(mavariable);  
5     }  
6 }
```

Dans ce cas-ci, la variable entière `mavariable` a été initialisée au sein du code Java et possède la valeur 8. C’est donc le nombre 8 qui sera affiché dans la console.

Vous pouvez remarquer que contrairement à l’affichage d’un texte, il n’y a pas de guillemets autour du nom de la variable lorsqu’on souhaite l’afficher.

Maintenant, si vous désirez obtenir à l’écran un texte sur plusieurs lignes, il faut adapter le `System.out.print` afin qu’il retourne à la ligne au moment voulu. En effet, si vous écrivez simplement plusieurs `System.out.print` les uns à la suite des autres, les chaînes de caractères seront affichées les unes à la suite des autres sans retour à la ligne.

Exemple 67 *Retour à la ligne*

```
1 public class start {  
2     public static void main(String[] args) {  
3         System.out.print("J’aime le Java!");  
4         System.out.print("Et aussi d’autres langages!");  
5     }  
6 }
```

Le résultat est donné à la Figure 27.

Pour insérer un retour à la ligne, il y a deux possibilités :

- Soit vous utilisez un caractère d’échappement `\n`.

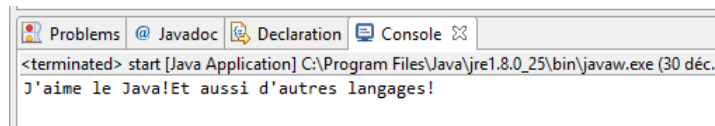


FIGURE 27 – Retour à la ligne - résultat

- Soit vous utilisez la méthode `println()` à la place de la méthode `print()`. Cette méthode a pour effet d’afficher le texte passé en paramètre dans la console suivi d’un retour à la ligne.

Note : En Java, les guillemets doubles sont des délimiteurs de chaînes de caractères. Si vous voulez afficher un guillemet double dans la sortie standard, vous devrez l’échapper avec un `\`.

8.6 Opérateurs de base

Les opérateurs de base utilisés en Java sont les suivants : `+` (addition), `*` (multiplication), `-` (soustraction), `/` (division), `^` (puissance), `%` (modulo). Ces instructions s’appliquent soit aux variables, soit directement à des valeurs. Le résultat de ces opérations pourra être affecté à des variables à l’aide du symbole d’égalité. Ce symbole sera précédé du nom de la variable qui recevra le résultat et suivi de l’expression à calculer.

Exemple 68 Opérateurs de base

```
1 int varEntier01 = 12;
2 int varEntier02 = 3;
3 varEntier01 = 2 * varEntier02;
```

Dans ce cas précis, la variable `varEntier01` prend la valeur 6 ($= 2 \times 3$).

8.6.1 Concaténation

Un autre opérateur qui nous sera bien utile dans ce cours est l’opérateur de *concaténation* de chaînes. Il s’utilise pour accoler des chaînes de caractères (des fragments de texte) soit avec d’autres chaînes de caractères, soit avec des variables contenant des nombres. Cet opérateur correspond au signe `+`. Une concaténation n’aura lieu que si la variable de stockage est une chaîne de caractères (`String`), ou peut être convertie en `String` "facilement". Le `+` est l’équivalent du `.` en PHP.

Exemple 69 Concaténation

```
1 public class start {
2     public static void main(String[] args) {
3         int ma_variable = 8;
4         String mon_texte = "Mon résultat est ";
5         mon_texte = mon_texte + ma_variable;
6         System.out.print(mon_texte);
7     }
8 }
```



```

7 | }
8 | }

```

Dont le résultat est donné à la Figure 28.

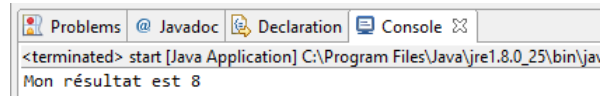


FIGURE 28 – Résultat Concaténation

Exemple 70 Concaténation 2

```

1 | public class start {
2 |     public static void main(String[] args) {
3 |         String mon_texte = "Simon";
4 |         mon_texte = "Bonjour " + mon_texte ;
5 |         System.out.print(mon_texte);
6 |     }
7 | }

```

Dont le résultat est donné à la Figure 29.

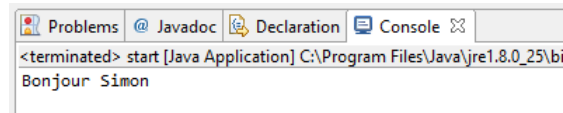


FIGURE 29 – Résultat Concaténation 2

8.6.2 Cast de variables

Attention :

Très important : on ne peut faire des opérations arithmétiques que sur des variables de même type ! On évitera par exemple de diviser un `int` par un `float` ! Ceci est valable pour tous les opérateurs arithmétiques et pour tous les types de variables numériques.

Afin d'effectuer les opérations arithmétiques proprement selon cette règle, il est possible d'effectuer des *casts* de variables, ou encore des *conversions d'ajustement*. Par exemple, imaginons que nous voulions effectuer une opération proprement entre les deux variables de type différents suivantes :

```

1 | int i = 123;
2 | float j = 2.5;

```

Dans ce cas, nous allons effectuer une conversion d'ajustement de la variable `i` d'un type `int` à un type `float`, c'est à dire que nous allons convertir la variable `i` en `float`. Cela se fait en écrivant

```

1 | float k = (float)i;

```

La variable `k` a toujours la même valeur que la variable `i` mais est cette fois de type `float`.

Inversément, nous pourrions effectuer une conversion d'un type `float` à un type `int`, avec toutefois une perte de précision dans ce cas.

```
1 | int l = (int)j;
```

La variable `l` contient la valeur 2 : la valeur de la variable `j` a été tronquée par la conversion et seule la partie entière est stockée.

Il est également possible de caster le résultat d'une opération mathématique en la mettant entre parenthèses et en la précédant du type souhaité. Par exemple, nous pouvons forcer le résultat à être un résultat entier :

```
1 | double nbre1 = 3, nbre2 = 2;
2 | int resultat = (int)(nbre1 / nbre2);
3 | // resultat=1 (perte de précision)
```

Que se passe-t-il si on fait l'inverse : déclarons deux entiers et stockons le résultat dans un `double`.

```
1 | int nbre1 = 3, nbre2 = 2;
2 | double resultat = (double)(nbre1 / nbre2);
3 | System.out.println("Le résultat est = " + resultat);
```

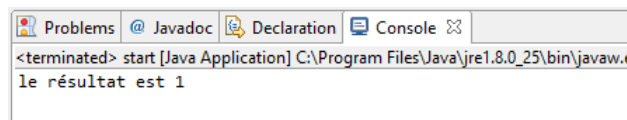


FIGURE 30 – Conversion - résultat

Le résultat obtenu est donné à la Figure 30. Ce résultat étonnant s'explique par le fait que, dans le langage Java, il y a la notion de priorité d'opération. L'affectation, le calcul, le cast,... sont des opérations et Java les fait dans un certain ordre. Dans le cas qui nous intéresse, il y a trois opérations : un calcul, un cast sur le résultat du calcul et une affectation dans la variable `resultat`. Le code est exécuté dans cet ordre. Le compilateur fait d'abord le calcul (ici $3/2$), le convertit en `double`, puis l'affecte dans la variable `resultat`. Or, en Java, le résultat de la première opération, la division de deux `int` donne un `int` (ici 1). Ce résultat est ensuite converti (toujours 1 mais de type `double`), et la valeur est affectée à la variable de type `double` (encore et toujours 1).

Pour avoir un résultat correct sans perte de précision, il faudrait plutôt convertir chaque nombre avant de faire l'opération :

```
1 | int nbre1 = 3, nbre2 = 2;
2 | double resultat = (double)(nbre1) / (double)(nbre2);
3 | System.out.println("Le résultat est = " + resultat);
4 | // Le résultat affiché est 1.5
```

Notons qu'il est aussi possible de convertir une variable numérique en chaîne de caractère et vice-versa.

Exemple 71 *Conversion d'un int en String*

```
1 | int i = 12;  
2 | String j = new String();  
3 | j = j.toString(i);
```

j est une variable de type String contenant la chaîne de caractères 12.

Exemple 72 *Conversion d'un String en int*

```
1 | int i = 12;  
2 | String j = new String("12");  
3 | int k = Integer.parseInt(j);
```

k est une variable de type int qui contient le nombre 12.

Il existe des équivalents à `Integer.parseInt()` pour les autres types numériques : `Double.parseDouble()` transforme du texte en variable de type double (nombres contenant des virgules).

8.7 Récupération des données saisies par l'utilisateur

Dans cette section, nous allons stocker des informations entrées par la saisie clavier dans des variables afin de pouvoir les utiliser et les manipuler ensuite. Ceci est un peu le penchant en Java des formulaires en PHP : nous allons inter-agir avec le programme. Pour ce faire, nous devons utiliser l'objet **Scanner**. Cet objet peut prendre différents paramètres, dans cette section nous n'en n'utiliserons qu'un seul, celui qui correspond à l'entrée standard (console).

Lorsque nous voulons afficher dans la console, nous utilisons la commande `System.out.print()`; c'est à dire que nous appliquons la méthode `print()` sur la sortie standard `System.out`. Ici, pour récupérer ce que l'utilisateur a entré dans la console, nous allons utiliser l'entrée standard `System.in` et l'objet **Scanner**. Toutefois, avant de pouvoir utiliser l'objet **Scanner**, il faut indiquer à Java où il peut trouver la classe correspondant à cet objet. Les classes, en Java, sont regroupées dans des *packages*. Un package est un ensemble de classes. En fait, c'est un ensemble de dossiers et de sous-dossiers contenant une ou plusieurs classes. En ce qui concerne la classe **Scanner**, elle se trouve dans le package `java.util`. Pour pouvoir utiliser cette classe, il est nécessaire d'importer ce package dans votre projet Java. La syntaxe utilisée pour importer ce package est présentée à la Figure 31.

Maintenant que la classe **Scanner** a été importée, il est possible de récupérer les données saisies dans la console. Détaillons, étape par étape et à partir de la Figure 32, ce que vous devez écrire dans votre code.

1. La première ligne encadrée est l'appel à la classe/objet **Scanner** avec l'entrée standard `System.in`. Notez que le nom donné `source` (on aurait pu choisir tout autre nom valide) sera employé dans la suite pour récupérer les données de l'entrée standard.

```
*start.java ☒  
1 import java.util.*;  
2 public class start {  
3  
4     public static void main(String[] args) {
```

FIGURE 31 – Code

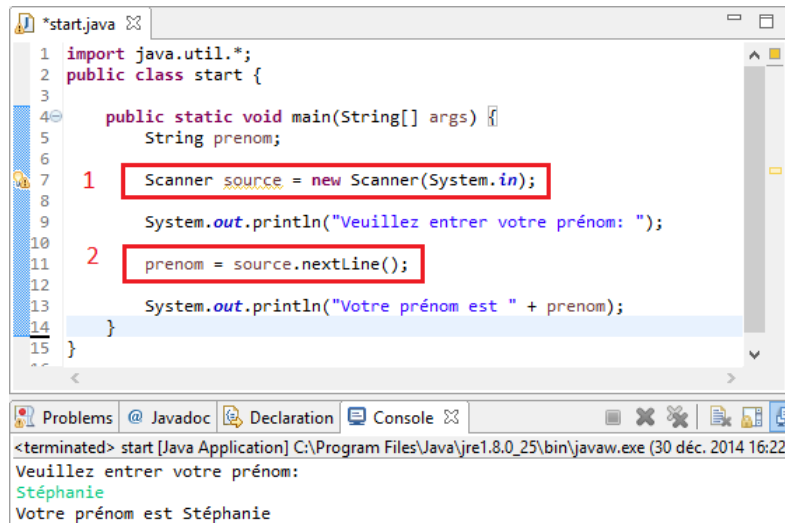


FIGURE 32 – Code

- Après avoir affiché un texte dans la console demandant à l'utilisateur d'entrer son prénom, à l'aide de la méthode `println` que nous connaissons déjà, la ligne d'instruction dans le second encadré permet de récupérer le texte saisi au clavier. Lorsque cette instruction est lue, Java attend que l'utilisateur entre une chaîne de caractères dans l'entrée standard (en vert dans la Figure 32) et appuie sur Enter. La méthode `nextLine()` récupère le contenu de toute la ligne saisie et replace la tête de lecture au début de la ligne suivante. Ensuite, cette chaîne de caractères est stockée dans la variable `prenom` de type `String` et pourra ensuite être affichée ou utilisée. Dans cet exemple, le prénom est ensuite affiché dans la console par exemple.

Le code précédent récupère une chaîne de caractères. Pour pouvoir récupérer un nombre, il suffit de remplacer `nextLine()` par `nextInt()`. De façon générale, pour récupérer un type de variable, il suffit d'utiliser la méthode `nextType` (en remplaçant `Type` par le type de variable désiré commençant par une majuscule). Cela n'est cependant pas valable pour le type `char`. La récupération d'un caractère seule se fait en récupérant toute la ligne avec la méthode `nextLine()` puis en allant chercher uniquement le premier caractère de la chaîne de caractères (voir plus loin).

Exemple 73 *Récupération d'un nombre*

```

1 public class start {
2     public static void main(String[] args) {
3         int nbre;
4         Scanner source = new Scanner(System.in);
5         System.out.print("Veuillez entrer votre âge: ");
6         nbre = source.nextInt();
7         System.out.print("Vous avez " + nbre + " ans");
8     }
9 }

```

Cet exemple fonctionne parfaitement pour autant que l'utilisateur entre un entier.

Attention :

Comme nous l'avons vu, la méthode `nextLine()` récupère le contenu de toute la ligne saisie et replace la tête de lecture au début d'une autre ligne. Par contre, si vous faite appel à une méthode comme `nextInt()` (ou `nextDouble()`,...) et que vous utilisez directement après la méthode `nextLine()`, celle-ci ne vous invitera pas à saisir une chaîne de caractères mais videra la ligne commencée par les autres instructions. En effet, comme les méthodes `nextInt` et `nextDouble` ne repositionnent pas la tête de lecture au début de la ligne suivante, l'instruction `nextLine()` récupère la suite de la ligne et le fait à leur place. Pour faire simple, le code suivant, ne permettra pas, contrairement à ce que nous pourrions croire, à l'utilisateur d'entrer une chaîne de caractères mais affichera directement "FIN!" après le texte "Saisissez une chaîne : ".

```

1 import java.util.Scanner;
2 public class start {
3     public static void main(String[] args){
4         Scanner source = new Scanner(System.in);
5         System.out.println("Saisissez un entier : ");
6         int i = source.nextInt();
7         System.out.println("Saisissez une chaîne : ");
8         String str = source.nextLine();
9         System.out.println("FIN !");
10    }
11 }

```

Pour éviter ce problème, nous prendrons donc l'habitude de vider la ligne après les instructions qui ne le font pas directement en ajoutant

```
1 source.nextLine();
```

directement après la ligne 6 du code précédent.

8.8 Instructions de test IF

Parfois, comme en PHP, on désire n'exécuter certaines instructions que si une condition particulière est satisfaite (par exemple, afficher "Bonjour" si l'heure actuelle est avant 18h, "Bonsoir" sinon). L'instruction Java permettant de tester une condition est le `if`. Si la condition est vraie, alors toutes les instructions entre accolades seront exécutées. Sinon, elles seront purement et simplement ignorées. Voici les différentes syntaxes possibles :

if	if...else	if...else if ... else
<pre> 1 if(cond){ 2 Instruct ; 3 }</pre>	<pre> 1 if(cond){ 2 Instruct1; 3 }else{ 4 Instruct2; 5 }</pre>	<pre> 1 if(cond){ 2 Instruct1; 3 }else{ 4 if(cond2){ 5 Instruct2; 6 }else{ 7 Instruct3; 8 } 9 }</pre>

8.8.1 Opérateurs de comparaison

Les principaux opérateurs de comparaison utilisés dans les tests sont repris dans le tableau ci-dessous. En d'autres termes, aucune différence avec le PHP !

Code	Résultat
<code>x == y</code>	retourne true si \$x est égal à \$y.
<code>x != y</code>	retourne true si \$x n'est pas égal à \$y.
<code>x > y</code>	retourne true si \$x est plus grand que \$y.
<code>x < y</code>	retourne true si \$x est plus petit que \$y.
<code>x >= y</code>	retourne true si \$x est plus grand ou égal à \$y.
<code>x <= y</code>	retourne true si \$x est plus petit ou égal à \$y.

Attention :

La comparaison de deux chaînes de caractères, contrairement au langage PHP, ne se fait pas à l'aide du double signe d'égalité mais à l'aide de la méthode `equals()`. Le code

```
1 | texte1.equals(texte2)
```

retourne true si les deux chaînes de caractères `texte1` et `texte2` de type `String` sont égales, false sinon.

8.8.2 Opérateurs logiques

Les opérateurs logiques sont repris dans le tableau suivant.

Code	Résultat
<code>condition1 AND condition2</code>	true si condition1 ET condition2 sont vraies.
<code>condition1 && condition2</code>	idem
<code>condition1 OR condition2</code>	true si condition1 est vraie OU condition2 est vraie.
<code>condition1 condition2</code>	idem
<code>!condition1</code>	true si condition1 n'est pas vraie

Exemple 74 Test

```

1 | public class start {
2 |     public static void main(String[] args) {
```

```

3   int nbre = 1;
4   if(nbre == 1) { // Si la variable vaut 1, on affiche
5       System.out.print("La variable vaut 1");
6   } else { // Sinon, on affiche
7       System.out.print("La variable est différente de 1");
8   }
9   }
10  }

```

Exemple 75 *Test 2*

```

1  public class start {
2      public static void main(String[] args) {
3          int nbre = 5;
4          if(nbre == 1){ // Si la variable vaut 1, on affiche
5              System.out.print("La variable vaut 1");
6          }else{
7              if (nbre == 2){ // Si la variable vaut 2, on affiche
8                  System.out.print("La variable vaut 2");
9              }else{ // Si la variable vaut autre chose, on affiche
10                 System.out.print("La variable ne vaut ni 1, ni 2");
11             }
12         }
13     }
14 }

```

Remarque :

Lorsque le code à l'intérieur des accolades n'est composé que d'une seule ligne, celles-ci deviennent facultatives. Néanmoins, dès que plusieurs instructions dépendent du `if` ou du `else`, il est impératif de les utiliser !

8.9 Les boucles

Si on désire exécuter le même ensemble d'instructions jusqu'à ce qu'une certaine condition soit remplie, on utilisera les boucles, comme en PHP. Différentes syntaxes existent. Les boucles les plus courantes sont `while` et `for`. Leur fonctionnement et leur syntaxe sont similaires aux instructions correspondantes en PHP.

8.9.1 Boucle while

La syntaxe de la boucle `while` est donnée par :

```

1  while (condition) {
2      Instruction1 ;
3      Instruction2 ;
4  }

```

Exemple 76 *Boucle while*

```

1  public class start {
2      public static void main(String[] args) {
3          int compteur = 0;

```

```
4   while (compteur < 4)
5   { // La boucle s'arrêtera lorsque la variable compteur sera
      égale à 4
6       System.out.println("Boucle numéro " + compteur);
7       compteur++; // Ajoute 1 au compteur à chaque passage de
          boucle
8   }
9 }
10 }
```

8.9.2 Boucle for

La syntaxe de la boucle `for` est donnée par :

```
1 for (compteur = min; compteur < max; compteur++) {
2     Instruction1 ;
3     Instruction2 ;
4 }
```

Exemple 77 Boucle for

```
1 public class start {
2     public static void main(String[] args) {
3         int compteur = 0;
4         for (compteur = 0; compteur < 4; compteur++)
5         { // La boucle s'arrêtera lorsque la variable compteur sera
              égale à 4
6             System.out.println("Boucle numéro " + compteur);
7         }
8     }
9 }
```

Comme en PHP le code `compteur++` est un raccourci pour `compteur =compteur+1`.

8.10 Les tableaux à une dimension

Il est possible de stocker plusieurs valeurs d'un même type derrière un seul nom de variable. La variable est alors un tableau (ou vecteur) composé de plusieurs cases mémoire, chacune numérotée et pouvant stocker une valeur. Notons que les cases seront numérotées à partir de 0 et pas à partir de 1.

Les tableaux peuvent contenir des nombres, des chaînes de caractères, etc. La déclaration se fait comme pour une variable de base mais on va utiliser en plus les crochets pour signaler et/ou dimensionner un tableau. Différentes syntaxes sont possibles.

Exemple 78 Déclaration avec affectation

```
1 public class start {
2     public static void main(String[] args) {
3         // Tableau d'entier
```



```

4 | int tabEntier[] = {0, 1, 2, 3, 4};
5 | // Tableau de double
6 | double tabDouble[] = {0.1, 0.2, 0.3, 0.4, 0.5};
7 | // Tableau de caractères
8 | char tabChar[] = {'a', 'b', 'c'};
9 | // Tableau de chaînes de caractères
10 | String tabString[] = {"Lundi", "Mardi", "Mercredi", "Jeudi", "
    Vendredi", "Samedi", "Dimanche"};
11 | }
12 | }

```

Exemple 79 Déclaration sans affectation

```

1 | public class start {
2 |     public static void main(String[] args) {
3 |         int tabEntier[] = new int [5];
4 |         double tabDouble[] = new double [6];
5 |         char tabChar[] = new char [3];
6 |         String tabString[] = new String [7];
7 |     }
8 | }

```

Dans ce cas-ci, les tableaux sont vides. Toutefois, contrairement au PHP, il est obligatoire de définir lors de leur déclaration le nombre de cases qu'ils contiennent. Par exemple, le tableau `tabEntier` contient 5 cases.

Une fois un tableau déclaré (avec ou sans affectation), il est possible d'affecter des valeurs à ces différentes cases. Pour ce faire, il suffit de préciser la case du tableau dans laquelle on souhaite insérer une nouvelle valeur. Notons que les variables affectées au tableau doivent posséder le même type que celui du tableau.

Exemple 80 Affectation dans un tableau déjà déclaré

```

1 | public class start {
2 |     public static void main(String[] args) {
3 |         int tabEntier[] = new int [5];
4 |         // Affectation de la valeur 12 dans la 3e case du tableau
5 |         // on commence à indexer à 0
6 |         tabEntier[2] = 12;
7 |     }
8 | }

```

L'utilisation d'une valeur d'un tableau se fait exactement de la même manière. On doit juste identifier l'élément du tableau que l'on souhaite utiliser.

Exemple 81 Parcours d'un tableau avec une boucle for et affichage des éléments

```

1 | public class start {
2 |     public static void main(String[] args) {
3 |         int tabEntier [] = {5, 10, 20, 30, 40, 50};
4 |         for(int index = 0; index < 6; index++){
5 |             System.out.println("Case no" + index + ", la valeur vaut "
    + tabEntier[index]);

```

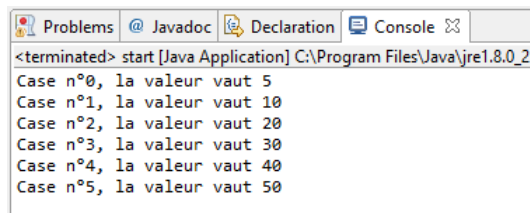


FIGURE 33 – Parcours d'un tableau - résultat

```

6   }
7   }
8 }

```

Le résultat de ce code est donné à la Figure 33.

Attention :

Les opérations arithmétiques sur des tableaux doivent s'effectuer élément par élément. Il n'est pas correct par exemple d'additionner deux vecteurs sans passer par les éléments.

Exemple 82 Addition de tableaux

```

1 public class start {
2     public static void main(String[] args) {
3         int tabEntier1 [] = {5, 10, 20, 30, 40, 50};
4         int tabEntier2 [] = {1, 2, 3, 4, 5, 6};
5         // Ceci est Faux !
6         tabEntier1 = tabEntier1 + tabEntier2;
7         //Ceci est correct:
8         for(int index = 0; index < 6; index++){
9             tabEntier1[index]= tabEntier1[index]+tabEntier2[index];
10        }
11    }
12 }

```

8.11 Les tableaux multidimensionnels

Les tableaux multidimensionnels (matrices quand le nombre de dimensions est 2) sont en fait des tableaux contenant des tableaux. Ce qui a été dit pour une seule dimension se généralise dès lors à autant de dimensions qu'on le souhaite. La déclaration d'un tableau à deux dimensions se fait de la manière suivante :

```

1 public class start {
2     public static void main(String[] args) {
3         // Declaration d'une matrice d'entiers de 3 lignes et 4
4         // colonnes.
5         int tabEntier [][] = new int [3] [4];
6     }
7 }

```

`tabEntier` est une matrice de 3 lignes et 4 colonnes. Plus exactement, `tabEntier` est un vecteur de 3 éléments. Chacun de ces éléments est lui-même un vecteur

de 4 entiers. Ainsi, le premier ensemble de crochets indique l'indice de la ligne et le second l'indice de la colonne. On pourra alors affecter des valeurs à ces éléments de la manière suivante :

```
1 tabEntier[0][0]=3;
2 // 1ère ligne, 1ère colonne, la valeur devient 3.
3 tabEntier[2][1]=4;
4 // 3ème ligne, 2ème colonne, la valeur devient 4.
```

N'oubliez pas que l'indexation des tableaux commence toujours à 0 et non à 1 ! Autrement dit, la case indexée [0] contient le premier élément du tableau, tandis que la case avec indexée [3] contient le quatrième élément.

Notons finalement qu'on peut, comme pour les tableaux à une dimension, déclarer et initialiser simultanément un tableau de la façon suivante :

```
1 // Matrice de deux lignes et trois colonnes
2 int tabEntier[][] = {{1,2,3}, {4,5,6}}
```

8.12 Méthodes utiles en Java

Le Java est un langage puissant qui propose de nombreuses méthodes intéressantes. Il serait impossible de les présenter toutes. En voici une brève sélection.

8.12.1 Mathématiques

Les méthodes suivantes font partie de la classe `Math` présente dans le package `java.lang`. Supposons disposer de la variable `nbr` de type `double`.

- `Math.abs(nbr)` : valeur absolue
- `Math.sin(nbr)` : sinus
- `Math.cos(nbr)` : cosinus
- `Math.tan(nbr)` : tangente
- `Math.pow(nbre, 2)` : élever à la puissance 2

8.12.2 Chaînes de caractères

Supposons disposer d'une variable `texte` de type `String`.

- `texte.length()` : donne la longueur de la chaîne de caractères
- `texte.toLowerCase()` : convertit la chaîne en minuscules
- `texte.toUpperCase()` : convertit en majuscules
- `texte.charAt(2)` : cette méthode rend le caractère placé à la 3-ième position (l'indexation commence à 0) de la chaîne de caractères.
- `texte.substring(3,10)` : extrait une partie de la chaîne (celle comprise entre les éléments d'index 3 à 10, le premier étant inclus et le dernier exclu)
- `texte.indexOf('t')` : explore la chaîne de caractère à la recherche de la sous-chaîne 't' et rend la position de sa première apparition.

- `texte.split(";")` : sépare la chaîne de caractères en utilisant le séparateur ";". Le résultat de cette séparation est un tableau dont chaque élément est une partie de la chaîne de caractères. De manière plus explicite, si la chaîne de caractères contient un seul séparateur (ex : "BMW; Volvo"), alors le résultat de `texte.split(";")` sera un tableau contenant 2 éléments. Si la chaîne contient deux séparateurs (ex : "BMW; Volvo; Ford"), alors le tableau contiendra 3 éléments, et ainsi de suite. Cette méthode sera très utile lors de la lecture de fichiers CSV.

8.12.3 Tableaux

La méthode `table.length` donne la taille du tableau `table`. Il n'y a pas de parenthèses au niveau de la méthode `length` quand on calcule la longueur d'un tableau (contrairement à ce qui se passe avec un `String`)

Exemple 83 Méthodes utiles

```
1 public class start {
2     public static void main(String[] args) {
3         String mon_texte01 = "J'AIME LE JAVA !";
4         String mon_texte02 = "hello !";
5         String mon_texte_temp;
6         int tab[] = new int[10] ;
7
8         mon_texte_temp = mon_texte01.toLowerCase();
9         System.out.println(mon_texte_temp);
10        // Résultat: j'aime le java !
11
12        mon_texte_temp = mon_texte02.toUpperCase();
13        System.out.println(mon_texte_temp);
14        // Résultat: HELLO !
15
16        int longueurTexte = mon_texte02.length();
17        System.out.println("Il y a " + longueurTexte + " caractères"
18        );
19        // Résultat: Il y a 7 caractères dans le texte "hello !"
20        // Remarque: les espaces blancs sont considérés comme des
21        // caractères
22
23        int longueurTableau = tab.length;
24        System.out.println("Longueur du tableau : " +
25        longueurTableau);
26        // Résultat: Le tableau tab[] a une longueur de 10
27    }
28 }
```

8.12.4 Aléatoire

Une façon de générer des nombres aléatoires est d'utiliser la classe `Random` contenue dans le package `java.util`. Une fois le package importé, il faut définir un objet `rand` (auquel on peut donner n'importe quel autre nom respectant les conventions de nommage) de type `Random` à l'aide de la syntaxe suivante. Cet objet pourra ensuite être utilisé pour générer plusieurs nombres aléatoires.

```
1 Random rand = new Random();
```

Il est possible d'ajouter un argument entre les parenthèses pour fixer la *seed*. Nous ne nous attarderons pas sur ce point mais sachez que la génération de nombres aléatoires par un ordinateur n'est pas purement arbitraire et soumise au hasard (il n'y a pas de hasard en informatique). Fixer la *seed* du générateur aléatoire aura pour effet que chaque fois que le programme sera exécuté avec la même *seed*, les nombres générés seront identiques d'exécution en exécution.

La génération de nombres aléatoires se fait ensuite avec une des syntaxes suivantes :

Exemple 84 Génération de nombres aléatoires

```
1 import java.util.*;
2 public class start {
3     public static void main(String[] args) {
4         Random rand = new Random();
5         int var1;
6         double var2, var3;
7
8         // Nombre entier entre 0 et bsup de manière uniforme
9         var1 = rand.nextInt(bsup);
10
11        // Nombre réel selon une loi de proba Gaussienne
12        var2 = rand.nextGaussian();
13
14        // Nombre réel entre 0 et 1 de manière uniforme
15        var3 = rand.nextDouble();
16    }
17 }
```

9 Lecture et écriture dans des fichiers

9.1 Exceptions

Avant d'apprendre à lire et écrire dans un fichier avec Java, nous devons voir préalablement une notion existant dans plusieurs langages de programmation : les exceptions. Une *exception* est une erreur conduisant le plus souvent à l'arrêt du programme.

Exemple 85 Exception - division par zéro

```
1 int j = 20, i = 0;
2 System.out.println(j/i);
3 System.out.println("coucou toi !");
```

Si vous créez une classe `main` avec pour seul code celui ci-dessus, vous obtiendrez un message d'erreur Java en rouge du type

`ArithmeticExceptionArithmeticException` et le programme s'arrêtera dès que l'exception sera rencontrée : le message "coucou toi !" n'est pas affiché .

Java contient en fait une classe nommée **Exception** dans laquelle sont répertoriés différents cas d'erreur. Le fait de gérer les exceptions s'appelle la *capture d'exception*. Ce traitement se fait via les blocs **try/catch**. Le principe consiste à repérer un morceau de code (par exemple, une division par zéro) qui pourrait générer une exception, de capturer l'exception correspondante et enfin de la traiter, c'est-à-dire d'afficher un message personnalisé et de continuer l'exécution. Les instructions susceptibles de lever une exception sont exécutées dans le bloc **try** et le bloc **catch** sert à définir des instructions qui seront exécutées en cas d'erreur.

Exemple 86 *Traitement d'exception - division par zéro*

Traitons par exemple l'erreur précédente en affichant un message personnalisé si une division par 0 est rencontrée.

```
1 public static void main(String[] args) {
2     int j = 20, i = 0;
3     try {
4         System.out.println(j/i);
5     } catch (ArithmeticException e) {
6         System.out.println("Division par zéro !");
7     }
8     System.out.println("coucou toi !");
9 }
```

Dans cet exemple, nous avons isolé le code susceptible de lever une exception dans le bloc **try**. Lorsque le programme atteint cette ligne, une exception du type **ArithmeticException** est levée. Le paramètre de l'instruction **catch** permet de préciser quel type d'exception doit être capturé. Ici, il s'agit d'un objet de type **ArithmeticException** que nous avons nommé **e**. L'exception étant capturée, les instructions du bloc **catch** sont exécutées, ce qui a pour effet d'afficher notre message personnalisé dans la console.

Le fait d'appeler l'exception **e** permet de s'en servir dans le bloc d'instructions **catch**. Nous aurions par exemple pu écrire

```
1 | System.out.println(e.toString());
```

à la place de la ligne 6, ce qui aurait eu pour effet d'afficher la nature de l'erreur (transforme l'exception **e** en une chaîne de caractères).

Remarque :

On aurait pu simplement déclarer une exception de type **Exception** en écrivant le code (**Exception e**) comme argument du **catch**. Cela aurait pour effet d'attraper **toutes** les exceptions levées dans le bloc **try**, quel que soit leur type. On peut également mettre plusieurs blocs **catch** qui se suivent afin de fournir un traitement spécifique pour chaque type d'exception. Un certain ordre doit cependant être respecté selon la hiérarchie des exceptions mais nous ne nous attarderons pas sur ce point.

L'avantage du principe de capture d'exception est que l'exécution du programme n'est pas interrompue : seule l'instruction dans le bloc `try` est interrompue. Dans l'exemple présenté, lorsque l'exception est capturée, le message spécifique est affiché puis le reste du programme, hors du bloc `try`, sera quand même lu. Autrement dit, il sera affiché *Coucou toi* dans la console malgré la rencontre d'exception précédente.

9.2 Récupérer les données contenues dans un fichier

Jusqu'à présent, l'utilisateur ne pouvait entrer des données dans son programme qu'en tapant directement ses données dans la console. Cette méthode est pratique pour de petits programmes. Mais dès qu'une grande quantité de données doit être entrée, cela peut poser divers inconvénients : erreurs d'encodage, multitude des données à entrer, perte de temps, ... en sont quelques exemples. Pour éviter ces inconvénients, il est possible de récupérer les données nécessaires au programme à partir d'une autre source telle que les fichiers textes, fichiers Excel, fichier CSV,... C'est ce qu'on appelle la lecture de fichier.

Dans l'exemple qui suit, un fichier de type CSV va être lu par Eclipse. Ce fichier (nommé `test.csv`) contient des nombres sur plusieurs lignes et répartis en 2 colonnes. Afin de le lire facilement avec Eclipse, ce fichier a été enregistré à la racine du projet Java en cours (Figure 35).

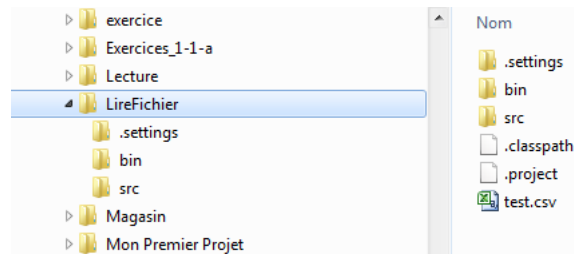


FIGURE 34 – Le fichier a été enregistré à la racine du projet Java en cours (dans ce cas-ci, le projet en cours s'appelle LireFichier)

Pour créer un fichier CSV (*Comma-Separated Values*), il suffit d'écrire des données dans une feuille Excel et de l'enregistrer dans le format CSV. Une fois créé, ce type de fichier est ouvert habituellement (et automatiquement) par Excel. Pour pouvoir voir le type de séparateur utilisé dans de tels fichiers, on peut les ouvrir grâce à Notepad++. Dans les captures d'écran ci-dessous, le fichier `test.csv` est ouvert par Excel et Notepad++. L'affichage du contenu diffère d'un logiciel à l'autre. Dans Excel, les nombres sont répartis sur 2 colonnes, tandis que dans Notepad++, les nombres sont séparés par des points-virgules qui représentent les colonnes.

Une fois le fichier créé, il est possible de récupérer son contenu dans Eclipse. La lecture d'un fichier est possible grâce à l'utilisation de méthodes apparte-

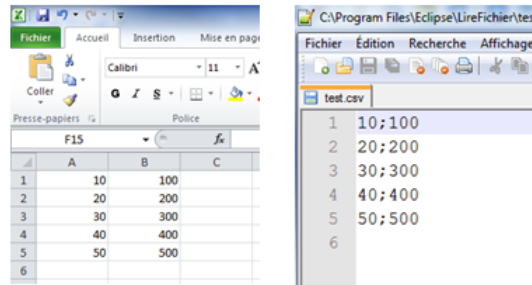


FIGURE 35 – Fichier test.csv ouvert dans Excel (à gauche) et dans Notepad++ (à droite)

nant au package `java.io.*`. Pour pouvoir les utiliser dans un projet Java, il est nécessaire d'importer le package comme cela a déjà été fait précédemment pour l'objet `Scanner`. Une fois le package importé, les méthodes qu'il contient deviennent accessibles au sein du projet et vont permettre de lire le contenu du fichier.

Dans l'exemple important qui suit, le fichier `test.csv` va être lu et son contenu va être stocké au sein d'un tableau d'entiers à 2 dimensions. Il est important de bien comprendre **chaque ligne** du code fourni ci-dessous.

Exemple 87 Lecture d'un fichier csv

```

1  import java.io.*;
2  public class MainLireFichier {
3      public static void main(String[] args) {
4
5          /* Création d'un tableau d'entier à deux dimensions */
6          int tabMulti[][] = new int[2][100];
7          int cpt = 0;
8          try{
9              /* Ouverture en mode lecture du fichier */
10             BufferedReader monBeauFichier = new BufferedReader(new
                FileReader("test.csv"));
11             String ligne ;
12
13             /* Tant que le fichier contient des lignes, on boucle */
14             while ((ligne = monBeauFichier.readLine()) != null)
15             {
16                 /* Découpe le String en utilisant le séparateur ; */
17                 String[] ar = ligne.split(";");
18
19                 /* Transformation des String en int
20                 et stockage dans le tableau d'entiers */
21                 tabMulti[0][cpt] = Integer.parseInt(ar[0]);
22                 tabMulti[1][cpt] = Integer.parseInt(ar[1]);
23                 cpt++;
24             }
25             /* Fermeture du fichier */

```



```
26     monBeauFichier.close();
27 }
28 catch (Exception e){
29     System.out.println(e.toString());
30 }
31
32 /* Affichage du tableau d'entiers */
33 int i = 0;
34 int j = 0;
35
36 /* On boucle autant de fois qu'il y a de lignes dans le
   fichier */
37 while(i < cpt) {
38     j = 0;
39     /* On boucle autant de fois qu'il y a de colonnes dans
       le fichier */
40     while(j < 2){
41         System.out.print(tabMulti[j][i] + "|");
42         j++;
43     }
44     System.out.println();
45     i++;
46 }
47 }
48 }
```

Détaillons les instructions utilisées dans le code de lecture de fichier ci-dessus.

1. `BufferedReader monBeauFichier = new BufferedReader(new FileReader("test.csv"));`

Cette instruction permet d'ouvrir le fichier en mode *lecture*. La variable créée se nomme `monBeauFichier`. Le nom réel du fichier doit être placé dans la dernière parenthèse. Dans notre exemple, le nom du fichier est `test.csv`.

2. `while ((ligne = monBeauFichier.readLine()) != null){...}`

Cette instruction lit l'ensemble du fichier ligne par ligne. L'instruction s'arrête lorsque toutes les lignes auront été lues. Ou, en d'autres mots, pour traduire littéralement la condition, lorsque la ligne du fichier lue est vide (ceci est représenté par la valeur `null`). Chaque ligne du fichier lue est placée dans une variable de type `String` nommée `ligne` déclarée précédemment.

3. `String[] ar = ligne.split(";");`

A chaque passage dans la boucle, la variable `ligne` de type `String` contient chaque fois une ligne du fichier lu. Dans notre exemple, chaque ligne du fichier CSV est composée de plusieurs nombres séparés par des points-virgules (ex : première ligne = 10; 100). Afin de pouvoir séparer ces nombres et les stocker dans des variables séparées, on utilise la méthode `split()` vue précédemment. Le séparateur employé est ici le point-virgule. A chaque fois qu'un point-virgule est trouvé, un nouvel

élément est créé dans le tableau `ar[]`.

4. `tabMulti[0][cpt] = Integer.parseInt(ar[0]);`
 - Lorsqu'on lit les lignes d'un fichier, celles-ci sont considérées comme du texte et doivent impérativement être stockées dans une variable de type `String`. C'est pourquoi, les variables `ligne` et `ar[]` vues précédemment étaient de ce type.
 - Néanmoins, dans notre exemple, les données qu'on souhaite récupérer du fichier sont des nombres, et même pour être plus précis, ce sont des entiers (variable de type `int`).
 - Ces nombres sont pour l'instant stockés dans le tableau `ar[]` de type `String`. Ils sont donc considérés comme du texte et non pas comme des entiers. Afin de pouvoir utiliser ces nombres dans le code (et donc pouvoir faire des calculs, ...), il faut les transformer en entiers. C'est possible grâce à la méthode `Integer.parseInt` vue précédemment.

Remarquons que les blocs `try/catch` sont ajoutés afin de capturer les exceptions éventuelles (par exemple si le fichier à lire n'est pas trouvé,...) et afficher le type d'exception rencontré le cas échéant.

9.3 Ecrire des variables dans un fichier

L'écriture de données dans un fichier correspond à la démarche inverse de la lecture. En effet, dans le cas de la lecture, on lit le fichier et on stocke son contenu dans des variables (qui sont vides au départ) au sein d'Eclipse. Tandis que dans le cas de l'écriture, on crée des variables dans Eclipse, on leur assigne des valeurs au sein du programme, et puis seulement on stocke leur contenu dans le fichier.

Dans l'exemple qui suit, le contenu d'un tableau à 2 dimensions (`tabIntMulti[][]`) va être stocké dans le fichier `ecriture.csv`. Afin de respecter la structure du tableau, 3 colonnes vont être créées dans le fichier grâce au séparateur point-virgule et à des retours à la ligne (`\n`).

Exemple 88 *Ecriture dans un fichier*

```

1  import java.io.*;
2
3  public class MainEcrireFichier {
4      public static void main(String[] args) {
5          /* Création d'un tableau d'entier à deux dimensions dont le
6             contenu va être stocké dans le fichier */
7          int tabIntMulti[][] = {{10,20,30},{11,21,31},{12,22,32}};
8
9          /* Ecriture du tableau dans le fichier: 3 colonnes */
10         String adresseFile = System.getProperty("user.dir")+"\\
11             ecriture.csv";
12         /* user.dir : Spécifie le répertoire du projet en cours */
13         /* ecriture.csv : nom du fichier et son extension */

```

```

13     try{
14         /* Ouverture en mode écriture du fichier */
15         BufferedWriter monBeauFichier = new BufferedWriter(new
            FileWriter(adresseFile));
16
17         for(int cpt = 0; cpt < tabIntMulti.length ; cpt++)
18         {
19             /* Ajout des points-virgules pour marquer la séparation de
                colonnes */
20             /* Ajout de \n afin de marquer le retour à la ligne */
21             monBeauFichier.write(tabIntMulti[cpt][0] + ";" +
22                 tabIntMulti[cpt][1] + ";" +
23                 tabIntMulti[cpt][2] + "\n");
24         }
25         /* Fermeture du fichier */
26         monBeauFichier.close();
27     }
28     catch (Exception e){
29         System.out.println(e.toString());
30     }
31 }
32 }

```

Détaillons les instructions utilisées dans le code d'écriture de fichier.

1. `String adresseFile =`
`System.getProperty("user.dir")+"\\ecriture.csv";`
 — L'instruction `System.getProperty("user.dir")` permet de récupérer automatiquement le chemin d'accès menant jusqu'au répertoire courant du projet en cours. Par exemple, si le projet a été enregistré sur le disque C, dans le dossier Eclipse, dans le sous-dossier `EcrireFichier`. Alors le résultat de cette instruction sera le suivant :
`"C:\Eclipse\EcrireFichier"`.
 — A la suite de ce chemin d'accès, on ajoute encore par concaténation le nom du fichier dans lequel nous allons écrire. Remarquons que le nom du fichier est précédé de deux backslashes. En réalité, ce que le programme lira sera le chemin d'accès suivant :
`"C:\Eclipse\EcrireFichier\ecriture.csv"`. Toutefois, puisque le backslash a une signification particulière en Java, il est nécessaire de le doubler.
 — Ce chemin d'accès complet est stocké dans la variable de type `String adresseFile`.
2. `BufferedWriter monBeauFichier =`
`new BufferedWriter(new FileWriter(adresseFile));`
 Cette instruction permet d'ouvrir le fichier en mode *écriture*. La variable créée se nomme `monBeauFichier`. Le chemin d'accès du fichier doit être placé dans la dernière parenthèse (ici `adresseFile`).
3. `for(int cpt = 0; cpt < tabIntMulti[0].length ; cpt++){...}`
 Cette instruction `for` permet de parcourir les éléments du tableau à 2 dimensions `tabIntMulti [] []`. Pour connaître automatiquement la taille

de la deuxième dimension de ce tableau, la méthode `length` vue précédemment est utilisée. Ainsi, on demande à la méthode `length` de calculer le nombre d'éléments du tableau (ici 3).

4. `monBeauFichier.write(...)`

A chaque passage dans la boucle cette instruction permet d'écrire dans le fichier ouvert en mode écriture. La chaîne de caractères écrite correspond aux éléments de la ligne indexée `cpt` séparés par des points-virgules et terminée par un retour à la ligne `\n`.

Attention :

On ne peut pas ouvrir un même fichier en lecture et en écriture simultanément. Il faut d'abord fermer le fichier en lecture pour l'ouvrir en écriture et vice-versa.