

TEX

Ceci est un titre

30 janvier 2020

 Oreste de savoir

La connaissance pour tous et sans pépins

Table des matières

Introduction	5
Contenu masqué	7
 I. Créer vos premiers documents	 8
1. LaTeX et les traitements de texte	10
1.1. Qu'est-ce qu'un traitement de texte	10
1.1.1. Préparation à la mise en page	10
1.1.2. Format de fichier	10
1.2. Différents types de traitements de texte	11
1.2.1. Traitements de textes WYSIWYG	11
1.2.2. Traitements de textes WYSIWYM	12
1.2.3. Traitements de textes à balisage	12
1.3. Présentation de LaTeX	13
1.3.1. Histoire	13
1.3.2. LaTeX	14
1.3.3. Qui utilise LaTeX et pourquoi ?	15
 2. Installation et premiers pas	 17
2.1. Installation	17
2.1.1. TeX Live, Windows	17
2.1.2. Tex Live, Linux	18
2.1.3. MacTeX sous OS X	19
2.1.4. Un éditeur de texte	19
2.2. Les principe généraux de LaTeX	20
2.2.1. Les commandes, la base de LaTeX	20
2.2.2. Les <i>packages</i> , un système d'extensions	21
2.2.3. Production d'un premier fichier PDF	22
2.3. Le compilateur	23
2.3.1. Déroulement de la compilation	23
2.3.2. Les fichiers créés	23
2.3.3. Plusieurs compilations ?	24
 3. Premiers codes	 25
3.1. Un premier document	25
3.1.1. Le préambule	25
3.1.2. La zone d'affichage	26
3.1.3. Notre seconde compilation	26
3.1.4. Les erreurs	28

3.2.	La rédaction en LaTeX	30
3.2.1.	Les paragraphes et l'écriture	30
3.2.2.	Des caractères réservés	31
3.2.3.	Les ligatures et l'espace insécable	33
3.2.4.	Les commentaires	33
3.3.	Le documentclass en profondeur	34
4.	Structurer un document	37
4.1.	Les commandes de hiérarchisation	37
4.1.1.	Des commandes bien utiles	37
4.1.2.	La table des matières	38
4.2.	Des documents français	40
4.2.1.	babel , un <i>package</i> international	40
4.2.2.	La coupure	41
4.2.3.	Des caractères supplémentaires	42
4.3.	Gestion du titre	42
4.3.1.	La commande \maketitle	42
4.3.2.	Page de titre	43
	Contenu masqué	44
5.	Quelques principes de TeX et de LaTeX	47
5.1.	Les tokens	47
5.2.	Les groupes	48
5.3.	Les environnements	49
5.3.1.	Définition d'un environnement	49
5.3.2.	Que font vraiment les commandes \begin et \end ?	50
6.	De la sémantique	52
6.1.	L'emphase	52
6.2.	Les listes	53
6.2.1.	Les listes à puces	53
6.2.2.	Les listes numérotées	54
6.2.3.	Les descriptions	55
6.2.4.	De l'utilisation des listes	55
6.3.	Les citations et les vers	55
6.3.1.	L'environnement quote	55
6.3.2.	L'environnement quotation	56
6.3.3.	L'environnement verse	57
7.	Annotations et références	58
7.1.	Notes	58
7.1.1.	Notes de bas de pages	58
7.1.2.	Notes de marges	60
7.2.	Le système de références	60
7.2.1.	Un système d'étiquettes	61
7.2.2.	Faire nos références	61
7.3.	Bien les utiliser	63
7.3.1.	Conseils d'utilisations	63
7.3.2.	Ce n'est pas...	63

8. Différents types de documents	65
8.1. Des classes pour de plus gros documents	65
8.1.1. La classe <code>book</code>	65
8.1.2. La classe <code>report</code>	67
8.2. D'autres types de documents	67
8.2.1. Des lettres	67
8.2.2. Des présentations	69
8.3. Organiser ses fichiers	70
8.3.1. Structurer notre projet	70
8.3.2. La commandes <code>\input</code>	71
8.3.3. La commande <code>include</code>	72
8.3.4. Le <code>package</code> <code>subfiles</code>	73
 II. Compléter vos documents	 75
9. Des Mathématiques	77
9.1. Écriture de base	77
9.1.1. Les modes mathématiques	77
9.1.2. Syntaxe de base	78
9.1.3. Les commandes usuelles	78
9.2. Symboles	82
9.2.1. Les lettres	82
9.2.2. Opérateurs binaires	83
9.2.3. Relations binaires	84
9.2.4. Logiques et ensembles	85
9.2.5. Délimiteurs	86
9.2.6. Flèches	87
9.3. Environnements mathématiques et autres commandes utiles	89
9.3.1. Environnements	89
9.3.2. Théorèmes	91
9.3.3. Autres commandes utiles	92
 10. De nouvelles commandes	 94
10.1. Créer ses commandes	94
10.1.1. Créer une commande	94
10.1.2. Créer un environnement	96
10.1.3. Redéfinition de commande et d'environnements	96
10.2. Le CTAN, découverte de classes et de packages	98
10.3. Utiliser les documentations	99
Contenu masqué	100
 11. Mise en forme	 102
11.1. Le texte	102
11.1.1. Changement de taille	102
11.1.2. Changement de fontes	103
11.1.3. Choix de police	105
11.2. Alignement du texte	106

11.3. Couleur	107
11.3.1. Du texte en couleur	107
11.3.2. Définir ses couleurs	108
12. Images, tableaux et texte préformaté	110
12.1. Images	110
12.1.1. Le package <code>graphicx</code>	110
12.1.2. Modifier l'image	111
12.1.3. Mode brouillon	112
12.2. Tableaux	112
12.2.1. Composer un tableau	112
12.2.2. Modifier le tableau	115
12.3. Texte préformaté	118
12.3.1. La commande <code>\verb</code>	119
Contenu masqué	119
13. Les flottants	121
13.1. Les flottants	121
13.1.1. Qu'est-ce qu'un flottant ?	121
13.1.2. Utiliser les flottants	121
13.1.3. Légendes et références	122
13.2. Placer les flottants	123
13.2.1. Les options de placement	123
13.2.2. Des légendes sans flottants	124
Conclusion	126

Introduction

Vous rêvez d'apprendre le LaTeX ? Vous voulez produire des documents esthétiques qui respectent les règles typographiques ? Ou encore pouvoir écrire des mathématiques facilement dans vos documents ? Vous êtes au bon endroit. Ce tutoriel permet d'apprendre les bases du LaTeX de manière à pouvoir poursuivre son apprentissage tranquillement.



Prérequis

Savoir vous servir d'un ordinateur devrait suffire...

Objectifs

Apprendre les bases du LaTeX.

Être capable de produire un document structuré.



À qui s'adresse ce tutoriel ?

Comment faut-il suivre ce tutoriel ?

Ce tutoriel s'adresse à toute personne souhaitant apprendre le LaTeX, voire renforcer ses connaissances en LaTeX. Il peut donc être suivi par quelqu'un qui a déjà quelques bases en LaTeX.

Ce tutoriel s'attache à donner une bonne compréhension de ce qu'est LaTeX et de ce que nous faisons lorsque nous l'utilisons. Pour cela, certaines notions seront vues en profondeur, et l'accent sera mis sur les « bonnes pratiques » et les astuces utiles. Le but du lecteur n'est pas de tout apprendre, mais de bien comprendre, quitte à ce qu'il doive dans un premier temps s'aider du tutoriel (ou de n'importe quel autre tutoriel et voire même d'un mémo) pour rédiger un document.

De plus, certains passages de ce tutoriel sont mis dans une balise secret. Ce sont des passages qui présentent des fonctionnalités pas forcément utiles au premier regard. Ils sont là pour le lecteur curieux ou qui a déjà quelques bases. Le lecteur débutant peut tout à fait les sauter et revenir dessus plus tard ou lorsqu'il en aura envie (voire jamais). Bien sûr, le lecteur débutant peut aussi lire ces passages par curiosité ou parce qu'ils lui serviront.

Ces passages se présenteront de la manière suivante : en gras, la raison du bloc secret et ce qui sera présenté suivi du bloc secret.

Ici, nous allons voir comment se présentent les blocs secrets.

☞ Contenu masqué n°1

Nous sommes maintenant prêts à nous lancer dans la mystérieuse aventure...



À venir

Ce tutoriel est toujours en rédaction. D'autres chapitres et sections suivront.

Contenu masqué

Contenu masqué n°1

Ils se présentent de la manière suivante.

[Retourner au texte.](#)

Première partie

Créer vos premiers documents

I. Créer vos premiers documents

Si vous avez cliqué sur cette partie, c'est que l'apprentissage du LaTeX vous intéresse. Dans cette section du tutoriel, nous apprendrons les bases nécessaires à la création de tous documents, ainsi que les bonnes pratiques à privilégier et les erreurs à éviter. Sans plus vous faire attendre, vous pouvez passer à la suite !

1. LaTeX et les traitements de texte

Avant de commencer à produire des documents avec LaTeX, faisons un petit tour d'horizon sur les traitements de texte, leur fonctionnement et sur LaTeX.

1.1. Qu'est-ce qu'un traitement de texte

1.1.1. Préparation à la mise en page

Comme leur nom l'indique, les logiciels de traitement de texte traitent du texte afin de les préparer à la mise en page. Pour ce faire, le logiciel a besoin que l'on ait donné une **signification** aux mots du texte, qu'il y ait une **sémantique**. La **sémantique** est ce qui fait qu'un ou plusieurs mots ou des paragraphes ne sont plus seulement des mots mais représentent quelque chose, par exemple un titre, une mise en évidence (emphase), du code source. La sémantique est ensuite traduite visuellement.

Ainsi, un traitement de texte **n'est pas fait pour écrire du texte**. C'est bien entendu possible, mais fortement déconseillé. C'est le rôle de l'éditeur qui, s'il est bien configuré, est très agréable à utiliser pour saisir du texte. Il faut donc le choisir avec soin. Cette séparation du fond et de la forme est moins un obstacle qu'un avantage. Saisir son texte séparément permet de se concentrer sur son contenu, sa sémantique, et pas sur sa forme.

Malheureusement, les traitements de texte graphiques (basés sur un langage de balisage) sont moins adaptés à cette séparation. Par exemple, avec Word, il est plutôt difficile d'identifier les mots importants dans un texte pour pouvoir décider de tous les mettre en gras ou en couleur par la suite. Il nous faudra tous les mettre en gras à la main, ce qui peut être extrêmement fastidieux.

1.1.2. Format de fichier

Pour enregistrer le travail effectué, on utilise des fichiers. Chaque traitement de texte ou presque utilise un format spécifique. On peut néanmoins les classer en deux grandes catégories : les formats dits **simples**, qui sont lisibles (ce qui ne veut pas dire compréhensibles) et éditables avec un éditeur de texte, et les formats dits **complexes** qui ne peuvent être lus que par le logiciel de traitement de texte.

1.1.2.1. Les formats simples

On utilise principalement deux formats : **XML** et ses dérivés (par exemple **HTML**) et le texte brut (qui comprend les langages de balisage légers et le langage LaTeX). **XML** est très lourd à éditer à la main (mais très facilement compréhensible pour un logiciel) alors que c'est l'inverse pour le texte brut.

Les formats simples sont surtout utilisés dans par les traitements de textes à balisage et ceux dits **WYSIWYM**, qui sont des logiciels graphiques faisant apparaître la sémantique.

Les formats simples sont pérennes, le texte brut ne dépendant de rien d'autre que son encodage (manière d'enregistrer les caractères) qui, s'il est bien choisi, sera lisible partout.

Le traitement automatisé d'un fichier simple est aisé par rapport aux fichiers complexes.

1.1.2.2. Les formats complexes

Surtout utilisés par les logiciels **WYSIWYG** comme Word et LibreOffice et aussi par les **WYSIWYM**, ils contiennent toutes les informations liées au document : le texte, bien sûr, mais aussi les éventuelles images ainsi que la mise en page.

Non ouvrables avec un éditeur de texte, ces formats ne sont pas pérennes pour peu qu'ils ne soient pas libres (c'est à dire que leurs spécifications sont accessibles pour tous).

Le traitement automatisé de tels formats est difficile.

1.2. Différents types de traitements de texte

Il y a trois grandes familles de logiciels de traitement de texte : les **WYSIWYG**, les **WYSIWYM** et ceux basés sur un balisage du texte. Tous ont leurs qualités et leurs défauts.

1.2.1. Traitements de textes WYSIWYG

Les traitements de textes de type **WYSIWYG** sont les plus courants. Ce sont généralement des logiciels complets, par exemple Word ou LibreOffice Writer. L'éditeur de texte, la mise en page et les correcteurs sont étroitement liés.

Ces logiciels s'utilisent visuellement, c'est à dire que le résultat final est visible à l'écran. Il existe donc une tentation de faire la mise en page en même temps que la saisie, ce qui est une perte de temps et génère des erreurs. Ainsi, le fond et la forme sont liés, d'autant que la sémantique est invisible (il faut connaître l'aspect que donne le logiciel aux éléments sémantiques et distinguer ce qui est réellement un élément sémantique d'une « reproduction »).

Il y a une différence fondamentale d'utilisation de ce type de logiciels suivant le secteur d'activité. Ainsi, le grand public s'en sert pour saisir le texte, créer la sémantique (ce qui n'est pas systématique, par ignorance) et mettre en page le texte. Un traitement de textes n'étant pas spécialisé dans la mise en page, il en résulte un document au mieux médiocre. Le monde de l'édition utilise des logiciels de mise en page spécifiques (par exemple Adobe InDesign,

I. Créer vos premiers documents

QuarkXpress et Scribus) qui, s'ils peuvent là aussi être utilisés comme traitements de textes (ce qui est ardu), ne sont utilisés que pour la mise en page. Le texte est sémantisé avec un logiciel de traitement de texte puis importé.

La manière correcte d'utiliser un traitement de texte graphique est de séparer les tâches. On saisira d'abord le texte, **puis** on appliquera la sémantique, et alors seulement on mettra en page.

1.2.2. Traitements de textes WYSIWYM

Ces logiciels sont rares. Ils sont là-aussi complets, comme LyX. Contrairement aux précédents, les traitements de textes **WYSIWYM**, on ne visualise pas le rendu final, mais la sémantique du document. La mise en page est générée automatiquement une fois l'aspect des différents éléments défini.

1.2.3. Traitements de textes à balisage

Terminons par les traitements de textes utilisant un langage de balisage. Ce ne sont pas des logiciels complets comme les **WYSIWYG** et les **WYSIWYG**. L'éditeur de texte est à choisir soi-même, et il est conseillé de pouvoir colorer le texte en mettant en évidence la syntaxe du langage (ceci n'est qu'une aide visuelle qui n'existe que dans l'éditeur : les fichiers seront toujours bruts). La mise en page est générée automatiquement en prenant en compte les réglages de l'utilisateur. Utiliser un correcteur nécessite d'en trouver un indépendant de tout logiciel.

Pour faire apparaître la sémantique, on utilise des **balises**. Voici un exemple d'emphase en **HTML** :

1	Ceci est du texte normal. <code></code> Cette phrase est mise en évidence. <code></code>
---	---

`` signifie « début de l'emphase » et `` « fin de l'emphase ».

On ne s'occupe plus du rendu de cette mise en évidence (qui peut devenir de l'italique, du gras, ou tout autre chose). C'est le mécanisme utilisé par tous les traitements de textes en interne. Lorsqu'on sélectionne une portion de texte et qu'on y applique un style, le logiciel insère des balises autour du texte sélectionné.

Comme on travaille sur du texte brut, on peut ajouter la sémantique au fur-et-à-mesure de la saisie, mais il est préférable d'avoir rédigé le document en entier au brouillon.

Il existe un certain nombre de langages de balisage. Ils ont chacun une utilisation propre. Il y a ainsi les langages lourds, basés sur **XML** (comme le **HTML**), fastidieux à utiliser à la main et presque jamais utilisés pour le traitement de textes. Les langages légers comme Markdown et Textile sont faits pour être utilisés à la volée. Très peu verbeux, ils sont discrets et peuvent presque être lus directement. À titre d'exemple, voici une emphase en markdown :


```
1 Ceci est *très* important.
```

Cet exemple se passe de commentaires. Ces langages quasi-naturels sont utilisés pour rédiger. On utilise ensuite souvent des convertisseurs vers, par exemple, [HTML](#) ou LaTeX.


Les langages intermédiaires sont plus verbeux que Markdown ou Textile mais plus digestes que les dérivés de [XML](#). LaTeX est un langage intermédiaire, comme Lout (qui lui ressemble beaucoup).

1.3. Présentation de LaTeX

1.3.1. Histoire

Dans les années 70, un informaticien du nom de [Donald Knuth](#)  trouvait que les logiciels à sa disposition pour écrire du texte avec des formules mathématiques n'étaient pas assez bons. Il décide alors (avec l'aide d'autres personnes) d'écrire le sien, qui sortira en 1978 sous le nom de TeX. Le but était de fournir un outil simple à utiliser pour composer des textes d'une grande qualité typographique. TeX comporte environ 300 commandes. On les appelle des **primitives**.

Voyant que certaines fonctions étaient compliquées à utiliser, Donald Knuth complète TeX en créant des commandes pour faciliter son utilisation. Le langage ainsi obtenu (combinaison de TeX et de commandes) est appelé un **format**. Il s'agit du format *Plain TeX*.

Malgré cela, le format Plain TeX reste compliqué et les utilisateurs ont commencé à écrire des extensions pour simplifier la composition des textes et éviter de tout réécrire depuis zéro. Au début des années 80, [Leslie Lamport](#)  fusionne un grand nombre d'extensions afin d'obtenir un langage de plus haut niveau : LaTeX. On utilise maintenant la deuxième version de LaTeX, LaTeX2e, sortie en 1994. LaTeX3 est en développement depuis le début des années 90.



Il ne faut pas confondre TeX, le langage, et le moteur, tex, qui est le programme chargé de transformer le code en document.

1.3.1.1. Anecdotes

- TeX, en tant que création de Donald Knuth, a la réputation d'être parfait. Il est même dit que le dernier bogue aurait été corrigé le 27 novembre 1985.
- Le numéro de version de TeX tend vers ∞ . Sa dernière version, sortie en 2008, est la 3.1415926.
- Le « X » de LaTeX est en fait un chi grec et se prononce donc comme un K.
- On désigne les utilisateurs de LaTeX sous le nom de TeXniciens (soit « techniciens »).

1.3.2. LaTeX

LaTeX est, comme nous l'avons dit, un ensemble de commandes destinées à faciliter l'utilisation de TeX. En gros, ce n'est qu'un sur-ensemble de TeX. Ainsi, le mode mathématique de LaTeX est celui de TeX. De même, la manière dont nous écrivons des paragraphes est héritée de TeX et ainsi de suite. Il ne faut pas oublier cela.

Tout comme TeX et tex, il ne faut pas confondre LaTeX et latex. En fait, latex est le programme chargé de transformer le code, et le **format**, LaTeX, qui est l'ensemble de commandes exécutées par le moteur. Nous écrivons notre code en LaTeX et le transformons grâce à latex.

1.3.2.1. Des outils divers

Pour utiliser LaTeX, il nous faut alors juste le moteur. Avec lui, nous pouvons transformer notre code en document. Cependant, nous pouvons nous équiper de nombreux autres logiciels. Conformément à la philosophie UNIX (« un programme fait une tâche et la fait bien »), latex se contente de la composition du document et les autres logiciels viennent le compléter. À ce titre, il y a des logiciels de gestion de bibliographie, ou encore de gestion d'index. Heureusement, il est généralement inutile de manipuler directement tous ces programmes, mais seulement une interface simple, mais faut-il encore savoir laquelle.

1.3.2.2. Plusieurs moteurs

Le moteur de TeX ne pouvait produire que des fichiers au format **DVI** et ne fonctionnait bien évidemment que pour TeX.

Le moteur actuel de LaTeX est PdfTeX qui permet de générer des fichiers au format **PDF**. Cependant, plusieurs autres moteurs sont ensuite apparus. Après PdfTeX, il y a eu XeTeX pour utiliser des polices *True Type* ou *Open Type*, c'est-à-dire celles utilisables directement par le système d'exploitation et ses logiciels) qui ne peut générer que des fichiers **PDF**. Enfin, il y a eu LuaTeX, basé sur le langage Lua, qui permet de réellement programmer à l'intérieur du document pour toujours plus de contrôle sur celui-ci. Il n'est pas encore finalisé à ce jour, mais est utilisable. LuaTeX est destiné à devenir le successeur de PdfTeX.

Un moteur peut avoir une commande pour créer un document **PDF** et une autre pour créer un document **DVI**. Cela donne lieu à plusieurs commandes pour créer un fichier. Voici un tableau récapitulant les compatibilités.

Format		TeX	PdfTeX	XeTeX	LuaTeX
Plain TeX	Sortie PDF	Oui	Oui	Oui	Oui
	Sortie DVI	Non	Oui	Non	Oui
LaTeX	Sortie PDF	Non	Oui	Oui	Oui
	Sortie DVI	Non	Oui	Non	Oui

Dans ce tutoriel, nous utiliserons le moteur PdfTeX et le format LaTeX. Passer à d'autres moteurs n'est pas très compliqué.

1.3.2.3. Des distributions

Toujours dans l'esprit UNIX, il n'existe pas une seule version de LaTeX (en tant que suite logicielle), il faut donc, là encore, choisir une distribution. Les distributions LaTeX sont identiques en pratique et contiennent tous les logiciels sus-mentionnés, bien que certaines disposent d'une interface simple pour installer des extensions. Les distributions les plus connues sont TeX Live, multiplateforme, MikTeX, pour Windows, et MacTeX pour OS X.

TeX Live étant disponible pour tous les systèmes d'exploitation, elle sera utilisée dans ce cours.

LaTeX est peu, voire pas du tout connu du grand public, ce qui a tendance à restreindre son utilisation à son domaine d'origine, c'est-à-dire les sciences.

1.3.3. Qui utilise LaTeX et pourquoi ?

Les mathématiciens et les informaticiens sont ceux qui utilisent le plus LaTeX (Donald Knuth étant informaticien). La physique compte beaucoup moins d'utilisateurs de LaTeX, mais les articles de recherche sont très souvent composés avec ce système, au moins par l'éditeur.

Très peu de particuliers et d'éditeurs utilisent LaTeX pour des documents non scientifiques. Les amateurs de typographie, lorsqu'ils ne peuvent s'offrir Adobe InDesign ou QuarkXPress, se tournent vers LaTeX. Les programmeurs amateurs, les geeks, aiment bien l'utiliser car il correspond à leur utilisation de l'ordinateur.

1.3.3.1. Avantages de LaTeX

1.3.3.1.1. Gagner du temps et se concentrer sur l'essentiel LaTeX, en séparant la forme du contenu, nous permet de nous concentrer exclusivement sur ce dernier, sans se prendre la tête et nous permet ainsi de gagner du temps lors de la rédaction.

1.3.3.1.2. L'intégration des mathématiques LaTeX est principalement utilisé par la communauté scientifique, et ce n'est pas par hasard : en effet, l'introduction de formules mathématiques dans le texte est une des forces de LaTeX puisqu'il permet de le faire à l'aide des commandes dont nous avons déjà parlé (écrire `a \in b` permettra par exemple d'obtenir $a \in b$).

1.3.3.1.3. La stabilité Comme dit plus haut, LaTeX et son ancêtre TeX sont réputés pour leur grande stabilité face aux bugs.

1.3.3.1.4. La gratuité LaTeX est également complètement gratuit et libre, ce qui en fait une excellente alternative gratuite aux autres traitements de texte, qui donne un rendu très professionnel.

1.3.3.2. L'inconvénient de LaTeX

Cependant, toute médaille a son revers, et celui de LaTeX est sa complexité. En effet, il est difficile pour un non-initié de l'utiliser, sans un apprentissage long et fastidieux. Mais le jeu en vaut la chandelle !

Maintenant que nous en savons un peu plus, nous pouvons commencer. Dans le prochain chapitre, nous installerons LaTeX.

2. Installation et premiers pas

Dans ce chapitre, nous installerons LaTeX et verrons quelques principes généraux à son sujet.

2.1. Installation

2.1.1. TeX Live, Windows

Sous Windows, nous allons utiliser la distribution [TeX Live](#) . Il y a plusieurs manières de l'installer.

2.1.1.1. Utiliser Internet

La manière la plus simple d'installer TeXLive est d'utiliser Internet. Il nous suffit d'aller sur [cette page](#) et de télécharger l'exécutable **install-tl-windows.exe**.

Sous Windows, il suffit de lancer l'exécutable téléchargé. On choisit alors « *Custom install* », avant d'appuyer sur « *Next* », puis sur « *Install* ».

Nous devrions maintenant avoir cette fenêtre sous les yeux.

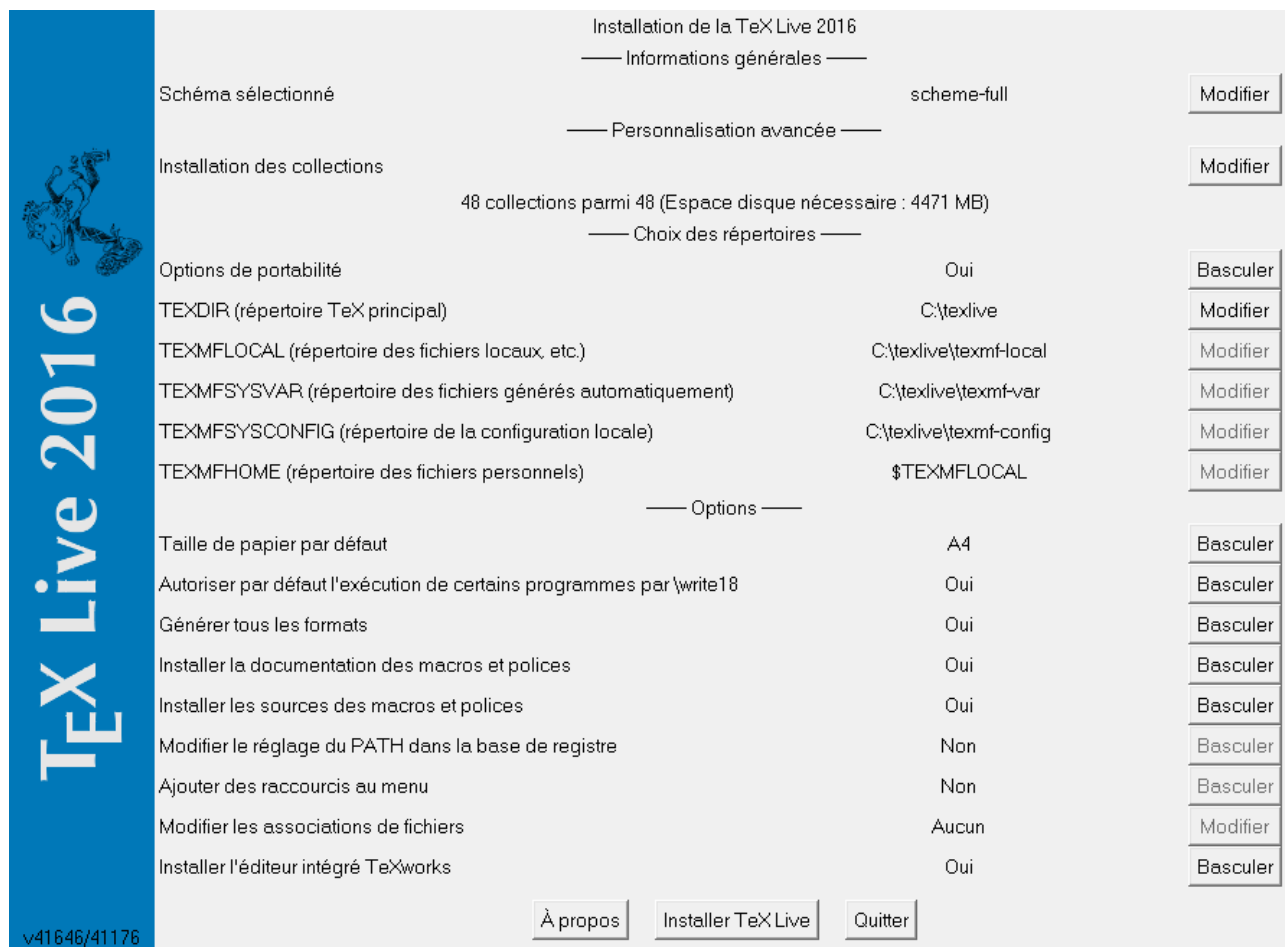


FIGURE 2.1. – Installation de TeX Live.

Nous conseillons de régler les options comme sur la capture d'écran (normalement, par rapport aux paramètres par défaut, il faut juste basculer « Options de portabilité » à oui). En particulier, choisir le schéma « *schema-full* » permet l'installation de tous les outils (cela installera également des outils que nous n'utiliseront peut-être pas).

Une fois les paramètres choisis, cliquons sur « Installer ». Les logiciels vont alors être téléchargés (cette opération peut être longue) et installés.

2.1.1.2. Télécharger une image disque complète

Si la méthode précédente ne convient pas, nous pouvons [télécharger une image disque de Tex Live](#) [↗](#). Il ne nous restera plus qu'à la graver ou à la monter. Une fois cela fait, il ne reste plus qu'à utiliser le CD ou l'image montée. L'installation se passe ensuite comme lorsque l'on utilise Internet.

2.1.2. Tex Live, Linux

Sous Linux, nous allons également installer TeX Live. Comme pour Windows, rendons-nous sur [cette page](#) [↗](#), mais cette fois, téléchargeons l'archive **install-tl-unx.tar.gz**.

I. Créer vos premiers documents

Décompressons le fichier téléchargé dans un nouveau dossier, rendons-nous dans ce dossier avec un terminal et exécutons la commande suivante.

```
1 $ ./install-tl --gui=perltk
```

Cette commande ouvre l'installateur en utilisant l'interface graphique `perltk`. Pour l'utiliser, `perl` et `Tk` doivent être installés (c'est généralement le cas).



Il est également possible d'utiliser son gestionnaire de paquets pour installer TeX Live (nous conseillons alors d'installer le paquet `texlive-full`).

2.1.3. MacTeX sous OS X

Sous OS X, nous allons utiliser la distribution [MacTeX](#). Il s'agit d'une adaptation de TeX Live pour OS X. Pour la télécharger, il suffit de se rendre sur [cette page](#) et de télécharger le fichier `pkg` qui est proposé.

2.1.4. Un éditeur de texte

Maintenant que nous avons installé une distribution, il nous faut choisir un éditeur de texte pour écrire. Il en existe beaucoup et il n'y en a pas vraiment de meilleur. Le meilleur éditeur que l'on puisse choisir est celui que l'on aime et qui nous convient.

Bien sûr, n'importe quel éditeur de texte pourrait suffire. Un simple logiciel du type bloc-note fait parfaitement l'affaire, mais il suffit d'avoir programmé un peu pour savoir qu'en fait, ils souffrent d'un grave défaut. Pour le mettre en évidence, comparons deux codes.

```
1 Voici des \emph{math} : $2 + 2 = 4$.
```


```
1 Voici des \emph{math} : $2 + 2 = 4$.
```

Le premier est plus lisible car certains mots sont mis en évidence grâce à de la couleur. On appelle cette fonctionnalité la **coloration syntaxique**. Elle permet de mettre en évidence certains éléments d'un langage et de cette manière aide à repérer les erreurs de syntaxe. Il nous faut choisir un éditeur qui propose la coloration syntaxique.


Le choix est ensuite complètement personnel. Certains éditeurs sont très simples à prendre en main, d'autres prennent un peu plus de temps à maîtriser mais se révèlent très puissants. Certains sont sobres et simples. D'autres encore sont très personnalisables. Il y en a pour tous les goûts et ce n'est qu'en essayant que l'on trouvera un éditeur satisfaisant.


I. Créer vos premiers documents

La plupart des programmeurs ont déjà leur éditeur favori. Ils peuvent le garder. Les autres pourront se renseigner sur ceux-là.

- [Notepad ++](#) , un éditeur libre et gratuit disponible sur Windows.
- Gedit, un éditeur libre et gratuit disponible sur Linux et intégré à Gnome.
- Kate, un éditeur libre et gratuit disponible sur Linux et intégré à KDE.

Les distributions TeX Live et MacTeX ont un éditeur de texte intégré, TeXworks que nous pouvons également utiliser. TeXworks a par exemple des boutons pour compiler le document plutôt que d'utiliser la ligne de commande.

Certains éditeurs facilitent la saisie de texte et de commandes LaTeX (notamment par des raccourcis). [TeXStudio](#)  par exemple est plutôt apprécié et offre des fonctionnalités intéressantes. Il permet notamment de compiler le document sans utiliser la lignes de commande, d'afficher le résultat directement, etc.

Notons que pour ne pas avoir à installer tous ces outils, nous pouvons utiliser LaTeX grâce à certains sites. Le site [Overleaf](#)  par exemple, permet d'écrire en LaTeX et d'obtenir le résultat. Ces outils ne sont pas aussi confortable qu'avoir une vraie distribution installée, mais ils peuvent être utiles si nous ne pouvons pas installer LaTeX sur un ordinateur ou si nous voulons tester un code rapidement.

2.2. Les principe généraux de LaTeX

2.2.1. Les commandes, la base de LaTeX

Les commandes¹ sont le cœur de LaTeX. Elles permettent par exemple d'indiquer que tel chose est un titre, qu'une autre est importante, etc.

Voici la syntaxe d'une commande.

1	<code>\commande</code>
---	------------------------

Une commande commence par un anti-slash² suivi de son nom. L'éditeur de texte choisi est censé mettre les commandes en évidence.

De plus, une commande peut prendre des arguments. Il y a deux titres d'arguments, les arguments obligatoires et les arguments optionnels. Par exemple, une commande `\titre` pourrait prendre en argument obligatoire le titre de notre document et en argument facultatif son sous-titre s'il y en a un.

On les indique de cette manière : les arguments optionnels (ou options) suivent le nom de la commande et sont placés entre crochets. Il peut également y en avoir plusieurs. Les arguments obligatoires, eux, sont entre accolades. Finalement voici la syntaxe générale d'une commande.

```
1 \commande[option]{argument}
```

Voyons maintenant des exemples de commande.

```
1 \commande
2 \commande{argument}
3 \commande[option]
4 \commande[option]{argument}
5 \commande[option]{argument}{autre_argument}
```

Dans l'ordre, nous avons :

- une commande sans argument ;
- une commande avec un argument obligatoire ;
- une commande avec une option ;
- une commande avec un argument obligatoire et une option ;
- une commande avec deux arguments obligatoires et une option.

2.2.2. Les *packages*, un système d'extensions

LaTeX en lui-même est très simple. Par exemple, à moins de l'écrire nous même, il n'existe pas de commande pour charger des images. Pour pallier à ce manque, nous pouvons étendre les fonctionnalités de LaTeX à l'aide d'extensions appelées *packages*.

Pour charger un *package*, on utilise la commande `\usepackage`. Elle prend en paramètre obligatoire le nom du *package* à charger. Certains *packages* ont des options, qu'on peut charger en passant des arguments facultatifs à la commande `\usepackage`.

```
1 \usepackage[option]{package}
```

Il est possible de charger plusieurs *packages* avec une seule commande. Pour cela, il suffit de séparer leurs noms par une virgule.

```
1 \usepackage{package, autre_package}
```

Cependant, il est déconseillé de le faire lorsque l'un des *packages* nécessite des options.



Certains *packages* sont dépendant d'autres *packages* ou changent le comportement d'autres *packages*. L'ordre dans lequel on les charge peut alors avoir une importance.

I. Créer vos premiers documents

Pour savoir dans quel ordre les charger, il faudra se référer à la documentation du *package*. Cependant, rapidement, on sait quel *package* fait quoi et de quel autre *package* il est dépendant. Pour avoir accès à la documentation d'un *package*, nous pouvons utiliser la commande `\texdoc nom_du_package`.

2.2.3. Production d'un premier fichier PDF

Nous allons maintenant créer notre premier fichier PDF. Pour cela, ouvrons notre éditeur de texte et copions le code suivant dans un nouveau fichier.

```
1 \documentclass{article}
2
3 \begin{document}
4
5 \end{document}
```

Pour le moment, nous ne sommes pas censés comprendre ce que tout ceci signifie, mais le voile sera levé au chapitre suivant.

Contentons-nous d'enregistrer ce fichier avec l'extension `.tex` (par exemple `premier_latex.tex`). Les fichiers LaTeX ont, par convention, cette extension. La plupart des éditeurs de textes comprennent à l'extension qu'il s'agit d'un fichier LaTeX et utilisent automatiquement la bonne coloration syntaxique.



Nous conseillons d'enregistrer le fichier dans un nouveau dossier. Lors de la création d'un fichier PDF, LaTeX génère un certain nombre de fichiers et c'est une bonne idée de mettre notre document dans un dossier à part.

Ensuite, il ne nous reste plus qu'à transformer le fichier `.tex` en PDF. Pour cela, ouvrons un terminal et rendons-nous dans le dossier où nous avons enregistré le fichier `tex` (grâce à la commande `cd`). Une fois ceci fait, il ne nous reste plus qu'à taper la commande `pdflatex <nom_fichier>`. Ainsi, si notre fichier s'appelle `premier_latex.tex`, nous allons taper la commande suivante.

```
1 pdflatex premier_latex.tex
```

La commande s'exécute et finalement, nous pouvons observer le résultat dans le fichier `premier_latex.pdf`.

-
1. Nous pourrions également croiser le mot « macro » à la place de « commande ».
 2. Aussi appelé barre oblique inversée, contre-oblique ou encore *backslash*.

2.3. Le compilateur

2.3.1. Déroulement de la compilation

La **compilation** est la transformation du fichier de code (en fichier PDF dans notre cas). La compilation est donc l'action que l'on effectue avec la commande `pdflatex`.

LaTeX est assez bavard et nous dit ce qu'il fait. Lors de la compilation de tout-à-l'heure, il nous a dit quelque chose du genre.

```
1 This is pdfTeX, Version 3.14159265-2.6-1.40.17 (TeX Live
   2016/W32TeX (preloaded format=pdflatex)
2 restricted \write18 enabled.
3 entering extended mode
4 (./premier_latex.tex
5 LaTeX2e <2016/03/31>
6 Babel <3.9r> and hyphenation patterns for 83 languages loaded.
7 (c:/texlive/texmf-dist/tex/latex/base/article.cls
8 Document Class: article 2014/09/19 v1.4h Standard LaTeX document
   class
9 (c:/texlive/texmf-dist/tex/latex/base/size10.clo))
10 No file premier_latex.aux
11 (./premier_latex.aux) )
12 No pages of output.
13 Transcript written on premier_latex.log.
```

Par exemple, en première ligne, il nous donne la version de TeX Live utilisée (ici c'est la version 32 bits de TeX Live 2016 sous Windows) et la commande de compilation utilisée.

2.3.2. Les fichiers créés

Lorsqu'on compile un document, plusieurs documents sont créés. Ils ont le même nom que le fichier et leur extension renseigne sur leur sujet.

- Le résultat de la compilation est le fichier `.pdf` (remarquons que la compilation de notre code n'en produit pas).
- Un fichier `.log` contient les informations de compilation. Sous Linux, les fichiers `.log` sont assez courants ; ils s'agit du journal des informations. Dans ce fichier, on trouve alors ce qui a été affiché dans le terminal lors de la compilation (et il peut même y avoir plus d'informations). Lorsqu'il y a eu une erreur de compilation, ce fichier est très utile, puisqu'il contient toutes les informations qui pourront nous aider à résoudre le problème (l'erreur rencontrée, la ligne de l'erreur, etc.)

D'autres fichiers comme le fichier `.aux` sont également créés.

2.3.3. Plusieurs compilations ?

Dans certains cas, plusieurs compilations sont nécessaires. Par exemple, lorsqu'on veut faire un index, il faut faire plusieurs compilations. Une première compilation répertorie les mots à placer dans l'index, et l'index est construit lors d'une compilation ultérieure. Dans ce cas, les informations nécessaires pour la seconde compilation (dans le cas de l'index, les mots à indexer) sont souvent placés dans le fichier `.aux` ou dans d'autres fichiers (dans le cas de l'index, il s'agit d'un fichier `.idx`).

En tout cas, nous devons retenir que plusieurs compilations sont parfois nécessaires à la construction de notre document.

Nous sommes maintenant prêts à vraiment démarrer, nous commencerons donc à écrire dès le chapitre suivant.

3. Premiers codes

Maintenant que nous avons installé LaTeX et vu un peu mieux son fonctionnement, il est temps de voir un premier document.

Dans ce chapitre, nous étudierons la base de la base. Nous verrons la structure d'un document et comment il est organisé.

3.1. Un premier document

Reprenons notre code précédent. C'est le squelette d'un document LaTeX.

```
1 \documentclass{article}
2
3 % le préambule
4
5 \begin{document}
6 % le corps du document
7 \end{document}
8 % rien ne doit être écrit ici
```

Après tout ce que nous avons déjà raconté, nous devons déjà reconnaître des commandes. Voyons à quoi elles correspondent.

3.1.1. Le préambule

Le préambule correspond à un espace avant notre document à proprement parler. Il est constitué de plusieurs lignes que nous allons dès maintenant analyser. Ici, notre préambule va de la première à la quatrième ligne. Le préambule sert à spécifier des informations à propos de notre document.

3.1.1.1. La classe de document

La commande `\documentclass` est généralement la première utilisée dans un document LaTeX. Cette commande sert à indiquer à LaTeX quelle classe de document utiliser. Une classe est en quelque sorte un modèle, un type de document. Chaque type de document a ses spécificités, sa mise en forme... La commande `\documentclass` permet d'indiquer quel type de document on veut écrire pour pouvoir disposer de ses spécificités justement. Il existe donc des classes pour faire des articles, des livres, des lettres ou encore des présentations.



La commande `\documentclass` peut alors être vue comme une commande qui permet de choisir un modèle de fichier.

Dans notre code, nous demandons à LaTeX d'utiliser la classe `article`. Il existe d'autres types de classe comme `book`, mais nous allons tout d'abord nous concentrer sur la classe `article`. D'où notre première ligne.

```
1 \documentclass{article}
```

3.1.2. La zone d'affichage

Nous avons dit que le préambule précède le document à proprement parler c'est-à-dire la zone dirigée vers l'affichage. En LaTeX, on rédige entre deux commandes : `\begin{document}` et `\end{document}`. La première commande sert à dire qu'on commence le document et la seconde qu'on a terminé le document.

Ce que nous écrirons avant `\begin{document}` ou après `\end{document}` ne sera affiché. En fait, si nous essayons de rédiger dans le préambule par exemple, nous obtiendrons une belle erreur et tout ce que nous plaçons après la commande `\end{document}` ne sera pas pris en compte et sera ignoré. On voit bien que tout ça est bien compartimenté :

- le préambule nous sert à indiquer les spécificités du document ;
- la zone d'affichage contient ce qui sera affiché.

Bien sûr, dans la zone d'affichage, on utilise également des commandes, par exemple pour indiquer qu'un mot est important.



Dans la zone d'affichage, nous n'écrivons pas que les choses à afficher. Nous pouvons par exemple définir une commande dans cette zone. C'est possible et cela se fait fréquemment, même si la plupart des commandes seront définies dans le préambule.

Il est donc faux de croire que dans la zone d'affichage on ne fait que rédiger. Il faut comprendre que ce qui doit être affiché doit être dans la zone d'affichage, pas que la zone d'affichage ne contient **que** ce qui doit être affiché.

3.1.3. Notre seconde compilation

Nous allons maintenant compléter un peu notre document. Écrivons ce que nous voulons dans la zone d'affichage et compilons le code ainsi obtenu. Compilons par exemple le code qui suit.

I. Créer vos premiers documents

```
1 \documentclass{article}
2
3 \begin{document}
4   Je suis un document.
5 \end{document}
```

Nous obtenons alors un document PDF d'une page avec juste écrit « Je suis un document ». Maintenant, compilons ce code.

```
1 \documentclass{article}
2
3 \begin{document}
4   Je suis un document créé grâce à la commande pdflatex.
5 \end{document}
```



Hé ! Les accents ne sont pas affichés !

Cela est dû à ce que l'on appelle l'[encodage](#) [↗](#). Selon le tutoriel passé en lien, L'encodage est « la façon de transcrire un texte grâce aux codes des caractères qui le composent, selon un jeu de caractères donné ». Il faut donc indiquer à LaTeX dans quel encodage notre fichier est enregistré. Il existe plusieurs encodages. Les encodages par défaut diffèrent suivants les systèmes, mais certains encodages sont, bien heureusement, disponibles sous tous les systèmes. Nous conseillons d'utiliser **UTF-8** partout. Il a l'avantage d'être disponible sur la majorité des systèmes et de gérer de nombreux caractères (en particulier les accents). Quasiment tous les éditeurs de texte proposent de choisir l'encodage voulu.

3.1.3.1. Les *packages* **inputenc** et **fontenc**

Il faut aussi que LaTeX sache dans quel encodage notre fichier est enregistré. Cela se fait grâce aux *packages*. Faisons un petit rappel à ce propos. Un *package* permet de rajouter de nouvelles fonctionnalités à notre code. Il existe des *packages* pour quasiment tout (pour faire des dessins par exemple). Les *packages* sont donc des extensions qui nous permettent de :

- rajouter des fonctionnalités ;
- modifier des paramètres ;
- corriger des problèmes.

Lorsque nous compilons avec la commande **pdflatex**, nous pouvons préciser un encodage grâce au *package* **inputenc** auquel on passe en option l'encodage voulu.



Nous devons faire attention au fait que si nous utilisons `inputenc` avec l'option `utf8`, notre fichier source doit effectivement être encodé en `UTF-8` pour ne pas avoir de mauvaise surprise.

Nous complétons encore cela avec le *package* **fontenc** qui lui se charge de la manière dont les fontes sont encodées et donc de la sortie obtenue. Nous allons l'utiliser avec l'option **T1** car elle contient l'essentiel des caractères des langues d'Europe de l'Ouest et permet la gestion des coupures de la majorité de ces langues. Nous complétons alors notre code.

```
1 \documentclass{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4
5 \begin{document}
6   Je suis un document créé grâce à la commande pdflatex.
7 \end{document}
```

Recompilons alors notre code précédent, nous remarquons que les accents sont maintenant présents.

3.1.4. Les erreurs

Si nous écrivons quelque chose qui ne plaît pas à latex, il nous retournera une belle erreur. Pour le moment, nous n'avons pas encore de quoi faire d'erreur, mais voyons déjà comment il réagit en cas d'erreur et que faire lorsque cela arrive.

Certaines erreurs bloquent la compilation, d'autres non. Une erreur non bloquante (aussi appelé avertissement) aura généralement un effet, même si elle n'empêche pas la création du document. Il vaut alors mieux essayer de faire un document sans avertissement qu'on ne comprend pas. Si on sait pourquoi on a un avertissement et qu'on connaît ses conséquences, on peut le négliger.



Il faut distinguer plusieurs erreurs. Les erreurs dues au moteur (la compilation du code pose un problème) et celles dues à LaTeX (le code n'est pas conforme au langage).

3.1.4.1. Une première erreur

Créons un fichier avec une erreur. Essayons par exemple de compiler le code précédent mais en écrivant `\begin{document`. Compilons donc le code suivant.

```
1 \documentclass{article}
2
3 \begin{document
4
5 \end{document}
```

Ici l'erreur est évidente. Il manque une accolade fermante après `\begin{document}`. La question est comment LaTeX nous le fera-t-il remarquer. Nous voudrions bien un beau message d'erreur qui dit explicitement qu'il manque l'accolade fermante après `\begin{document}`.

3.1.4.2. Les informations affichées lors de la compilation

Lorsque nous compilons le code précédent, nous obtenons dans la console ceci à quelques mots près.

```
1 This is pdfTeX, Version 3.14159265-2.6-1.40.17 (TeX Live
   2016/W32TeX (preloaded format=pdflatex)
2 restricted \write18 enabled.
3 entering extended mode
4 (./premier_latex.tex
5 LaTeX2e <2016/03/31>
6 Babel <3.9r> and hyphenation patterns for 83 languages loaded.
7 (c:/texlive/texmf-dist/tex/latex/base/article.cls
8 Document Class: article 2014/09/19 v1.4h Standard LaTeX document
   class
9 (c:/texlive/texmf-dist/tex/latex/base/size10.clo))
10 Runaway argument?
11 {document
12 ! paragraph ended before \begin was complete.
13 <to be read again>
14             \par
15 l.4
16
17 ?
```

Une erreur est toujours présentée de la manière suivante.

```
1 ! erreur
2 l. numéro de la ligne
```

3.1.4.3. Que faire en cas d'erreur

Lorsque nous obtenons une erreur, la première chose à faire est d'essayer de la comprendre. Avec notre code précédent, nous obtenons l'erreur `paragraph ended before \begin was complete.` détecté à la ligne 4 de notre document. Cela signifie que la primitive `\par` (qui correspond à deux retours à la lignes consécutifs) a été rencontrée dans l'argument de la commande `\begin`. On en déduit qu'on n'a pas fermé la commande `\begin` avant de sauter deux lignes.



Une primitive est une commande directement codée dans le moteur tex. Nous verrons plus tard pourquoi elle ne peut pas être présente dans la commande `\begin`.

Tout au long de notre apprentissage, nous verrons plusieurs messages d'erreurs classiques. Par exemple, nous obtenons l'erreur `Too many }'s` s'il y a trop d'accolades fermantes.

Une fois que nous avons compris l'erreur, il faut maintenant agir. Lorsqu'une erreur bloquante a lieu, le compilateur nous demande ce qu'on veut faire en utilisant un point d'interrogation. Nous pouvons faire plusieurs choix en indiquant une lettre. Il nous faut ensuite valider en appuyant sur Entrée.

- `q` ou `x` arrête la compilation (Nous pouvons également appuyer sur `Ctrl`+`D` pour cela. C'est sans doute la meilleure solution. Il nous suffit ensuite de corriger notre code avant de recompiler.
- `h` affiche de l'aide à propos de l'erreur rencontrée.

Nous pouvons également appuyer sur Entrée, auquel cas la compilation continuera. Notons cependant qu'une erreur en entraîne souvent d'autres et qu'il vaut mieux ne pas continuer à compiler un code qui contient des erreurs. De plus, la compilation s'arrête dans tous les cas au bout de cent erreurs.

3.2. La rédaction en LaTeX

3.2.1. Les paragraphes et l'écriture

La rédaction en LaTeX n'est pas compliquée. Si on fait abstraction des commandes, le bloc principal que nous allons utiliser est le paragraphe. Et il s'agit bien d'un bloc aussi bien dans notre code que dans le document `pdf`. Pour le rédacteur, un changement de paragraphe correspond à un changement d'idée (la plupart du temps).

Suivant ce que nous venons de raconter, écrivons un texte composé de plusieurs paragraphes.

```
1 \documentclass{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4
5 \begin{document}
```


I. Créer vos premiers documents

```
6 Je suis un document composé de plusieurs paragraphes. Voici un
  premier paragraphe.
7 Je suis allé à la ligne pour commencer un autre paragraphe.
8 Et maintenant un troisième paragraphe dans lequel il y a pleins
  d'espaces ici et là.
9 \end{document}
```

Compilons ce code et observons le résultat.



Quoi ! Nous n'obtenons pas de paragraphes ! Et nos espaces supplémentaires non plus ne sont pas là !

C'est dû aux règles de rédaction de Tex. Elles sont très simples, mais il faut les connaître. Voici les deux règles essentielles qui concernent les paragraphes.

- Un retour à la ligne n'est pas interprété comme un changement de paragraphe, mais comme une espace. Pour changer de paragraphe, il faut sauter une ligne.
- Sauter plusieurs lignes revient à sauter une ligne, donc à changer de paragraphes.

En fait, deux retours à la ligne consécutifs sont équivalents à la primitive `\par` (que nous avons déjà rencontré). Elle permet d'indiquer explicitement à Tex la fin d'un paragraphe.



Dans certains arguments de commandes, la primitive `\par` est interdite. La commande `\begin` est de ce type. On dit que c'est une commande « courte ». Avoir une commande courte permet d'être sûr que l'argument de la commande ne comporte pas plus d'un paragraphe (dans le cas contraire, on obtient une erreur, du même type que celle vue avec `\begin`).

Notons que de même, plusieurs espaces sont interprétées comme une seule espace.

Le comportement des paragraphes peut être changé, mais ce n'est pas l'objet de ce chapitre.

Il faut juste retenir ces deux règles qui sont vraiment essentielles. Ces règles peuvent paraître absurdes, mais il est logique si l'on pense en termes de séparation du fond et de la forme. Ce comportement nous permet de présenter notre code en laissant des espaces où l'on veut.

3.2.2. Des caractères réservés

Bon, essayons maintenant d'écrire un document avec quelques symboles.

```
1 \documentclass{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4
```

I. Créer vos premiers documents

```
5 \begin{document}
6   Dans mon document, je peux écrire de smileys :), ^^, :\ et même
   devenir riche $.
7 \end{document}
```

Et là, catastrophe, on obtient une erreur. Notre code ne compile pas.

Cela est dû au fait que certains caractères sont réservés. Cela veut dire que nous ne pouvons pas les taper tels quels dans le code pour les afficher dans le document final.

?

Pourquoi sont-ils réservés ? Et si on veut les utiliser alors ? Comment fait-on ?

Ces caractères sont réservés pour une bonne raison : ils signifient quelque chose. Ils permettent de coder une action. `\` par exemple permet de commencer une commande. Nous ne pouvons dès lors pas les écrire directement sans quoi le compilateur tentera de faire l'action que cette commande désigne. En écrivant `\ceci`, le compilateur tentera d'exécuter la commande `\ceci` et nous obtiendrons une erreur.

Pour obtenir ces caractères dans le texte, il nous faut utiliser des commandes. Voici un tableau des commandes à utiliser pour obtenir chaque caractère :

Caractère	Commande associée
{	\{
}	\}
%	\%
\$	\\$
&	\&
#	\#
~	\~
_	_
^	\^
\	\textbackslash

Comme nous pouvons le voir, il y a très peu de caractères réservés, et les commandes à utiliser pour les avoir dans le texte ne sont pas compliquées. Il y a des caractères dont nous ne connaissons pas encore l'utilité, mais nous allons les voir pendant le reste du cours.

3.2.3. Les ligatures et l'espace insécable

3.2.3.1. L'espace insécable

Nous ne sommes pas là pour des leçons de typographie, mais sachez que l'espace insécable `\hspace*` est une espace particulière qu'on place entre deux mots (ou signe de ponctuation) qui ne doivent pas être séparés par une fin de ligne. Il permet de les avoir tous les deux sur la même ligne. L'espace insécable s'emploie en français devant les signes de ponctuation doubles (points d'exclamation, deux-points, etc.) et en d'autres occasions.

En LaTeX, une espace insécable se fait avec le symbole `\hspace*`.

3.2.3.2. Les ligatures

En français, les ligatures `\oe` sont assez courantes (par exemple la ligature « œ » dans le mot œuf). LaTeX propose un certains nombres de ligatures et nous permet d'écrire ces mots.

Les ligatures comme « œ » et « æ » sont accessibles à l'aide des commandes `\oe` et `\ae` (les commandes à utiliser en majuscule sont `\OE` et `\AE`).

Ces ligatures donnent l'occasion de parler d'une spécificité des commandes sans argument : si une commande sans argument est suivie d'un espace, celui-ci n'est pas pris en compte. Ainsi, en écrivant `\oe uf`, on obtient « œuf » et non « œ uf ». Pour prendre en compte l'espace, il y a trois solutions :

- mettre une paire d'accolades vide après la commande : `\oe{} uf` ;
- mettre un antislash après la commande : `\oe\ uf` ;
- entourer la commande d'accolades : `{\oe} uf`.

La solution que nous croiserons le plus est celle de l'antislash.

D'autres ligatures sont automatiques. Par exemple, `---` et `--` permettent d'obtenir respectivement des tirets cadratins `\hrulewidth` et des tirets demi-cadratins `\hrulewidth`. Toutes ces petites choses nous facilitent grandement l'écriture.

3.2.4. Les commentaires

LaTeX possède une syntaxe pour les commentaires. Un commentaire commence par le signe `%` (nous comprenons pourquoi il est réservé) et se termine à la fin de la ligne. Les commentaires ne sont pas compilés et donc ne sont pas dans le document. Ils ne sont utiles que pour le lecteur du code.

Ils peuvent être utiles pour se rappeler l'utilité ou de la syntaxe de telle ou telle commande, ou encore pour ne pas compiler une partie du texte que nous ne sommes pas sûrs de vouloir dans notre document final. Nous pouvons par exemple commenter de cette manière.

```
1 \documentclass{article} % Utilisation de la classe article
2
```

```
3 % On charge des paquets pour l'encodage
4 \usepackage[utf8]{inputenc}
5 \usepackage[T1]{fontenc}
6
7 \begin{document}
8   Je suis un document créé grâce à la commande pdflatex.
9 \end{document}
```

Commenter est très utile, mais il ne faut pas non plus tomber dans l'excès et commenter chaque ligne. Il est conseillé de ne commenter que les lignes compliquées à comprendre.

3.3. Le documentclass en profondeur

La commande `documentclass` nous permet de définir le type de document que nous voulons. Pour cela, elle charge un *package*, le *package* `classe`. Tout comme `usepackage`, la commande `documentclass` peut prendre des options qui nous permettent de définir notre document encore plus finement. Un appel à `documentclass` ressemblera plus à cela.

```
1 \documentclass[option1, option2]{nom_de_la_classe}
```

Bien entendu, chaque classe a ses options. Il faut aller voir la documentation de la classe pour connaître les options qu'elle prend en charge. Pour voir la documentation des classes de bases, il nous faudra taper `texdoc classes`. Nous pouvons également [regarder ce site](#) (que nous verrons en détail plus tard dans le tutoriel). Vu que nous utilisons la classe `article`, voyons quelques-unes de ses options pour finir ce chapitre.

3.3.0.1. Taille de la police

La taille de la police dans le texte courant est la première option que nous verrons. Par défaut, cette taille est de 10 pt. La classe `article` nous laisse le choix entre trois tailles.

Taille (pt)	Option
10	10pt
11	11pt
12	12pt

Ainsi, pour avoir une taille de 12 pt, nous utiliserons cette commande.

```
1 \documentclass[12pt]{article}
```

3.3.0.2. Format de papier

Le format de papier est une option qui peut s'avérer intéressante dans le cas où nous voulons un document spécifique. Le papier par défaut est soit au format A4, soit au format letter (le format par défaut dépend de notre installation). Nous disposons néanmoins de plusieurs autres formats.

Format	Option
A4	<code>a4paper</code>
A5	<code>a5paper</code>
B5	<code>b5paper</code>
Letter	<code>letterpaper</code>
Legal	<code>legalpaper</code>
Executive	<code>executivepaper</code>

On a un joli paquet de format.



Le format par défaut est le format portrait. Pour utiliser le format paysage, il faut utiliser l'option `landscape`.

3.3.0.3. Recto verso

Nous pouvons aussi choisir entre un document recto verso et un document recto. La différence est qu'un document recto verso a des marges plus courtes sur les côtés qui seront contre la reliure, ceci afin d'avoir des marges à peu près identique lorsque le livre sera relié justement. LaTeX prend cela en charge grâce à deux options :

- l'option `oneside` permet d'obtenir un document en recto ;
- l'option `twoside` permet d'obtenir un document en recto verso.

La classe `article` est par défaut en recto simple. Si le document n'est pas destiné à être imprimé, l'option `oneside` est plus avantageuse.

3.3.0.4. Texte sur deux colonnes

Nous pouvons choisir de disposer le texte du document sur deux colonnes grâce à l'option `twocolumn`. L'option `onecolumn` permet au contraire d'avoir le texte sur une seule colonne (c'est l'option par défaut de la classe `article`).

Notons que l'option `twocolumn` peut être avantageusement remplacée par un *package*. Certains proposent même un plus grand choix de nombres de colonnes.

3.3.0.5. Document brouillon ou final

Nous pouvons choisir de compiler votre document en mode brouillon grâce à l'option `draft`. Le mode brouillon permet d'avoir une compilation un peu plus rapide. Pour cela, elle remplace par exemple les images par un cadre de la même taille. De plus, le mode brouillon permet de mettre en évidence des défauts tels que les dépassements de marge.

L'option par défaut n'est pas `draft` mais `final` qui permet d'obtenir le document final.

Bien sûr, nous n'avons pas besoin de retenir toutes ces options (la documentation est là pour ça). Généralement, nous utiliserons juste `12pt` et `a4paper` et laisseront le reste par défaut.

Avec ce chapitre, nous avons vu comment créer notre premier document, comment écrire et nous avons vu un début de personnalisation de notre document.

4. Structurer un document

Après avoir compilé notre premier document, il est temps de voir nos premières commandes qui s'utilisent dans le texte. Dans ce chapitre, nous apprendrons à donner du sens à certains mots, à mettre en évidence et nous verrons un *package* qui nous aide à avoir des documents dans la bonne langue.

4.1. Les commandes de hiérarchisation

Nous aimerions sûrement pouvoir découper votre document en chapitres, en sections, en sous-sections. Le sectionnement d'un texte est très important et permet d'organiser des idées et de ne pas perdre le lecteur. Il nous faut alors savoir que c'est l'une des choses qui se fait le plus facilement en LaTeX.

4.1.1. Des commandes bien utiles

LaTeX dispose en effet de quelques commandes pour nous aider dans cette tâche. Ces commandes correspondent à plusieurs niveaux de titres (chapitres, sections, sous-sections, paragraphes...) bien qu'on ne les utilise pas toujours suivant la signification de leur nom.

Le nombre de commandes est plus ou moins long en fonction de la classe que l'on a choisi. La classe `article` que nous étudions ne permet pas, par exemple, d'avoir de chapitre. Voici les commandes disponibles avec la classe `article` par ordre d'importance.

Niveau	Commande	Numérotation	Poids
Section	<code>\section</code>	1	1
Sous-section	<code>\subsection</code>	1.1	2
Sous-sous-Section	<code>\subsubsection</code>	1.1.1	3
Paragraphe	<code>\paragraph</code>		4
Sous-paragraphe	<code>\subparagraph</code>		5

Les commandes que nous utiliserons le plus sont `\section` et `\subsection`. La numérotation, la police, la taille et l'espacement sont gérés automatiquement. C'est l'une de ses forces.

Pour utiliser ces commandes, il suffit de les placer là où on veut dans le texte en leur passant comme argument le nom qu'on veut donner à la partie. Un exemple de document serait :

```
1 \documentclass[a4paper, 12pt]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4
5 \begin{document}
6   Le texte commence par un alinéa.
7   \section{Une première section}
8
9   \section{Une deuxième section avec une sous-section}
10    \subsection{Une première sous-section}
11    Le texte ne commence pas par un alinéa.
12 \end{document}
```

Compilons ce code. Nous pouvons y voir la numérotation et la différence de taille.

Ici, nous voyons comment changer le dernier niveau de sectionnement numéroté.

👁 Contenu masqué n°2

4.1.1.1. Annexes

Nous avons aussi la possibilité de créer une annexe. Pour cela, il nous faut utiliser la commande `\appendix`. Toutes les commandes `\section` qui seront utilisées après cette commande seront considérées comme faisant partie de l'annexe et seront donc numérotées différemment (des lettres A, B plutôt que des chiffres).

4.1.2. La table des matières

LaTeX nous permet de plus de construire une table des matières à partir des différentes parties introduites grâce aux commandes de sectionnement. Pour cela, il suffit d'utiliser la commande `\tableofcontents`. Cette commande place la table des matières à l'endroit où elle a été utilisée (donc utilisons la au début de votre document ou à la fin).

i

Pour faire apparaître la table des matières, le document doit être compilé deux fois. La première fois sert à lister les différentes parties, et la deuxième à créer la table.

Ici, nous voyons comment changer le dernier niveau de sectionnement qui apparaît dans la table des matières (par défaut, c'est la sous-sous-section).

👁 Contenu masqué n°3

4.1.2.1. Titres courts

Par défaut, le nom de partie qui apparaît dans la table des matières est le nom donné à la partie. Avec la commande `\section{Un nom de section vraiment très très, mais vraiment très long}`, le nom qui apparaîtra est « Un nom de section vraiment très très, mais vraiment très long ».

Ce serait fabuleux de pouvoir mettre un titre un peu plus court dans la table des matières et de malgré tout garder le titre complet dans l'article. Non seulement la table des matières sera plus lisible, et en plus la recherche sera facilitée. Pour faire cela, il suffit de passer à la commande de sectionnement le titre plus court en option :

```
1 \section[Nom court]{Un nom de section vraiment très très, mais  
   vraiment très long}
```

4.1.2.2. La variante étoilée

Il y a des cas où on peut vouloir une section et ne pas vouloir de numérotation. C'est le cas par exemple d'une section « Introduction » ou « Avant-propos ». Nous pouvons également gérer ces cas : chaque commande de sectionnement possède une variante étoilée qui ne numérote pas le titre et ne change pas la numérotation des titres de même niveaux. Par exemple, pour notre section « Avant-propos », on doit écrire ceci.

```
1 \section*{Avant-propos}
```



Si nous voulons supprimer la numérotation de tous les titres, il ne faut pas utiliser la variante étoilée à chaque fois. Nous verrons plus tard la manière propre de faire cela en redéfinissant la commande.

Cependant, lorsque l'on utilise la variante étoilée d'une commande de sectionnement, le titre n'apparaît pas dans la table des matières. Très embêtant pour nos introductions et autres avant-propos. Pour le faire apparaître, il faut utiliser la commande `\addcontentsline` après la commande étoilée. Cette commande prend trois arguments :

- le premier sera toujours `toc` pour « *Table Of Contents* » ;
- le deuxième est le niveau de titre ;
- le troisième est le titre que l'on veut faire apparaître.

Pour faire apparaître l'introduction, on utilisera donc ce code.

```
1 \section*{Introduction}  
2 \addcontentsline{toc}{section}{Introduction}
```

I. Créer vos premiers documents

Finalement, un document peut ressembler à ça.

```
1 \documentclass[a4paper, 12pt]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4
5 \begin{document}
6   \tableofcontents
7   \section*{Introduction}
8   \addcontentsline{toc}{section}{Introduction}
9   texte de l'introduction
10
11  \section{Première section}
12    texte d'introduction de la première section
13    \subsection{Sous-section 1}
14      texte de la sous-section 1
15    \subsection{Sous-section 2}
16      texte de la sous-section 2
17    \subsection{Sous-section 3}
18      texte de la sous-section 3
19
20  \section{Une deuxième section avec une sous-section}
21    texte d'introduction de la deuxième section
22    \subsection{Sous-section 1}
23      texte de la sous-section 1
24    \subsection{Sous-section 2}
25      texte de la sous-section 2
26
27  \section*{Conclusion}
28  \addcontentsline{toc}{section}{Conclusion}
29  texte de la conclusion
30
31  \appendix % On passe aux annexes
32  \section{Annexe 1}
33    texte de l'annexe 1
34
35  \section{Annexe 2}
36    texte de l'annexe 2
37 \end{document}
```

4.2. Des documents français

4.2.1. babel, un package international

Après avoir compilé les codes précédents, il y a quelque chose que nous n'avons pas pu manquer : le titre de la table des matières est « Contents », là où nous aimerions avoir « Table des matières ».

I. Créer vos premiers documents

Ce n'est pas trop gênant, mais nous aimerions quand même avoir des documents français.

Pour résoudre ce problème, nous allons utiliser le *package* **babel**. Il s'agit de l'un des principaux *packages* de LaTeX. Il a pour rôle de gérer les langues dans les documents. Il se charge donc de traduire certains mots, de modifier les règles de typographie, et il propose même des commandes pour faciliter l'écriture dans certaines langues.

Pour charger le *package* babel, il faut utiliser la commande `\usepackage` (comme pour tous les *packages*) en lui passant en option la ou les langues qui l'utilisent. Pour utiliser le français, nous utiliserons cette ligne.

```
1 \usepackage[french]{babel}
```

Avec l'option `french`, babel ajoute automatiquement les espaces insécables là où il le faut. Nous n'avons donc pas besoin de le faire manuellement.



babel agit sur beaucoup de *packages* en les mettant dans la bonne langue. Pour qu'il puisse agir sur tous les *packages* que nous avons, il faut le charger en dernier. En fait, les *packages* qui doivent être chargés après babel le spécifient.

Une autre méthode est de donner le langage à utiliser en option au `documentclass`. Les autres *packages* pourront alors détecter la langue utilisée et s'y adapter. C'est cette dernière méthode que nous allons adopter.

Le *package* babel offre de plus de nombreuses fonctions pour accorder notre document à la langue utilisée. Par exemple, la typographie française veut que les noms de famille soient écrits en petite capitale. babel nous donne la commande `\bsc` qui prend en paramètre un texte et se charge de le mettre en petites capitales. De plus elle met ce texte dans une boîte insécable ce qui veut dire que le nom ne pourra pas être coupé.

Ici, nous voyons comment gérer plusieurs langues avec babel.

☞ Contenu masqué n°4

4.2.2. La coupure

Ici, nous voyons comment nous pouvons gérer la coupure à la main.

☞ Contenu masqué n°5

4.2.3. Des caractères supplémentaires

En outre, en français, il y a des caractères indispensables. Si nous disposons d'un clavier qui nous permet de les utiliser, tant mieux. Sinon, nous avons plusieurs commandes et babel en rajoute. Avec babel, nous avons ces commandes :

- `\og` et `\fg` nous permettent d'obtenir les guillemets français « et » ;
- `\no`, `\nos`, `\No` et `\Nos` nous permettent d'obtenir « n° » et ses variantes majuscules et plurielles ;
- `\ier`, `\iere`, `\ieme`, `\iers`, `\ieres` et `\iemes` nous permettent d'obtenir «^{er}», «^{re}» et autres ;
- `\primo`, `\secundo`, `\tertio` et `\quarto` pour « 1° », « 2° », « 3° » et « 4° » et la commande `\frenchenumerate` avec un nombre en paramètre nous permet d'obtenir les autres nombres.

Puisque nous avons chargé un *package* pour gérer l'encodage **UTF-8**, nous pouvons écrire directement ces caractères dans notre document (si notre clavier nous le permet). Cependant, dans certains cas, c'est impossible et le seul moyen d'obtenir ces caractères est d'utiliser une commande.

De plus, avec ces commandes, on a automatiquement les espaces nécessaires, ce qui n'est pas forcément le cas lorsque nous écrivons le caractère nous même. Par exemple, en écrivant `\og Citation \fg`, les espaces insécables sont présents ce qui n'est pas le cas lorsque nous écrivons « Citation ». Pour régler ce problème, il nous faut rajouter dans notre préambule, après le chargement de babel, la commande `\frenchbsetup{og=«,fg=»}` qui indique que les caractères « et » correspondent à `\fg` et à `\og`. On peut alors écrire « Citation » et même «Citation» sans problème.

Ici, nous voyons comment écrire des lettres accentués grâce à des commandes (là encore, puisque que nous avons chargé un *package* pour gérer l'encodage **UTF-8** nous pouvons écrire ces caractères directement dans notre document).

👁 Contenu masqué n°6

4.3. Gestion du titre

Il est maintenant temps de se faire une page de titre.

4.3.1. La commande `\maketitle`

LaTeX nous propose la commande `\maketitle` pour faire des titres. Elle se charge de mettre en place le titre, d'écrire le nom de l'auteur, tout ça sans que nous n'ayons à nous en mêler. Cette commande est à utiliser juste après avoir écrit `\begin{document}` afin d'avoir le titre en première page.



Mais comment elle fait pour connaître le nom du titre et de l'auteur la commande `\make title` ?

En fait, les éléments du titre sont définis grâce à d'autres commandes à utiliser avant la commande `\maketitle` (elles peuvent être utilisées dans le préambule).

- La commande `\title` prend en paramètre le titre du document. Elle est obligatoire pour utiliser la commande `\maketitle`.
- La commande `\author` prend en paramètre les auteurs du document (s'il y en a plusieurs, il faut les séparer par la commande `\and`). Pour ne pas avoir d'auteurs, nous pouvons ne pas utiliser la commande `author`. Cependant, il vaut mieux transmettre à `\author` un argument vide (utiliser `\author{}`) pour ne pas avoir d'avertissements.
- La commande `\date` prend en paramètre la date du document. Nous pouvons lui passer ce que nous voulons. Pour avoir la date du jour, nous pouvons soit ne pas utiliser `\date`, soit lui passer `\today` comme argument. Pour ne pas avoir de date, il suffit de transmettre à `\date` un argument vide.



Nous pouvez utiliser la commande `\thanks` dans les commandes `\title` et `\author` pour obtenir une note de bas de page (elle prend en argument obligatoire la note à afficher). Elle est surtout utilisée dans les articles pour donner des informations complémentaires telles que l'adresse courriel.

4.3.2. Page de titre

Nous avons maintenant vu comment mettre en page le titre. En essayant de compiler un code avec la commande `\maketitle`, nous pouvons remarquer que le texte commence tout de suite après le titre. Nous pouvons changer cela et dédier une page au titre. Cela se fait grâce aux options du `documentclass` :

- l'option `titlepage` permet d'avoir le titre seul sur une page ;
- l'option `notitlepage` permet d'avoir le titre en haut de la première page.

Nous pouvons nous en douter, l'option par défaut pour la classe `article` est l'option `notitlepage`. Il ne tient qu'à nous de choisir l'autre option pour obtenir une page de titre.

Nous pouvons alors tester les deux options et observer le résultat.

```
1 \documentclass[french]{article} % Nous pourrions tester avec
   l'option titlepage
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \date{\today}
```

```
7 \author{\bsc{Karnaj} \and Zeste de Savoir}
8 \title{Test}
9
10 \begin{document}
11   \maketitle
12   Je suis un document dans lequel il y a un titre.
13 \end{document}
```

Ce chapitre nous aura permis d'apprendre à structurer votre document comme il faut et nous aura vu utiliser vraiment notre premier *package*. C'est dans ce chapitre et dans le suivant que nous verrons vraiment l'importance de la sémantique lorsqu'on utilise LaTeX.

Contenu masqué

Contenu masqué n°2

Pour changer le dernier niveau qui est numéroté, nous utilisons dans le préambule la commande `\setcounter` qui prend deux paramètres. La commande `\setcounter` permet de modifier la valeur d'un compteur, son premier paramètre est le compteur à modifier, la seconde est la valeur que l'on veut lui donner.

LaTeX utilise le compteur `secnumdepth` pour connaître le dernier niveau de sectionnement qui doit être numéroté. En changeant la valeur de ce compteur, on obtient alors le résultat que l'on veut. On va alors utiliser `\setcounter` avec ces deux paramètres.

- Le premier sera `secnumdepth` pour profondeur de la numérotation des sections.
- Le second sera le poids du dernier niveau de sectionnement à numéroté (c'est donc un chiffre entre 1 et 5 pour la classe `article`).

Ainsi, avec `\setcounter{secnumdepth}{4}`, on numérote jusqu'aux paragraphes.



Nous verrons les compteurs et leur utilisation plus tard. Ici, il nous faut juste savoir qu'ils existent.

[Retourner au texte.](#)

Contenu masqué n°3

Pour changer cela nous devons utiliser la commande `\setcounter` dans le préambule (comme pour changer le dernier niveau de numérotation). Cela signifie que nous allons changer la valeur d'un compteur.

Ici, ce sera la valeur du compteur `tocdepth` qui correspond à la profondeur de la « *Table Of Contents* ». Nous allons lui donner comme valeur le poids du dernier niveau que l'on veut faire

apparaître dans notre table des matières (ici, le deuxième argument de `\setcounter` est donc un chiffre entre 1 et 5 pour la classe article). [Retourner au texte.](#)

Contenu masqué n°4

Lorsque nous utilisons plusieurs langues, il faut utiliser la commande `selectlanguage` en lui passant en paramètre la langue que nous voulons utiliser pour changer de langue. Cette langue sera la langue active jusqu'à la prochaine commande `\selectlanguage`. La langue qui est active au début du document est celle qui avait été passé en dernier à babel (avec le code `\usepackage[langueA, langueB]{babel}`, ce sera `langueB`). [Retourner au texte.](#)

Contenu masqué n°5

Le moteur gère les coupures en essayant de couper au mieux les mots en fin de ligne et le *package* babel améliore la gestion des coupures. Cependant, il arrive que le moteur ne sache pas où couper un mot. Dans ce cas, il ne choisit pas et met tout le mot sur la même ligne. Testons par exemple ce code.

```
1 \documentclass[a4paper, 12pt, french]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \begin{document}
   ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
7 \end{document}
```

Tous les f sont sur la même ligne et ça dépasse de la marge droite. D'ailleurs, le moteur génère l'avertissement `Overfull \hbox` lors de la compilation. Cet avertissement signifie que quelque chose dépasse de la boîte (ici la page).

Heureusement, cela n'arrive pas tout le temps et surtout, si cela arrive, nous avons une commande pour préciser un endroit où un mot peut être coupé. Il s'agit de la commande `\-`. Il suffit de la placer là où le texte doit être coupé.



Le meilleur moyen de gérer les coupures reste quand même de laisser babel s'occuper de couper les mots comme il faut.

[Retourner au texte.](#)

Contenu masqué n°6

Nous disposons de plus de plusieurs commandes pour avoir les différents accents. Ces commandes sont à placer avant la lettre qui doit porter l'accent. Voici un tableau présentant les quelques accents les plus communs.

Résultat obtenu	Commande
è	\`e
é	\'e
ê	\^e
ë	\"e
ñ	\~n

[Retourner au texte.](#)

5. Quelques principes de TeX et de LaTeX

Dans ce chapitre, nous allons nous attarder un peu sur le fonctionnement de TeX et sur quelques principes qui nous seront utiles pour la suite.

5.1. Les tokens

Ici, nous allons voir comment TeX comprend en gros un fichier qu'on lui demande de compiler. En fait, un fichier en entrée est transformé en séquence de caractères et cette séquence de caractères est transformée en *tokens*, chaque *token* étant un caractère ou une séquence de contrôle (qui commence par `\`).

Les séquences de contrôles sont alors transformées en primitives (qui sont aussi des *tokens*). Les opérations correspondant à ces primitives sont alors effectuées ce qui nous permet d'obtenir une liste de pages qui est transformée dans le format demandé.

Bien sûr, ce que nous venons de décrire ne correspond que grossièrement au fonctionnement de TeX, mais cette description légère nous suffit.

Lorsque le fichier en entrée est lu, TeX a besoin de savoir à quoi chaque caractère correspond (pour savoir s'il s'agit d'une séquence de contrôle ou pas, etc.). Le code de catégorie (ou *catcode*) d'un caractère détermine le rôle de ce caractère. TeX attribue un code à chaque caractère qu'il lit. Il y a un code pour les lettres, un autre pour les espaces, etc. Cela permet par exemple de distinguer les caractères spéciaux.

Nous pouvons alors comprendre grossièrement de quelle manière TeX fonctionne. Voici comment le livre « TeX pour l' impatient » le présente en utilisant une métaphore très imagée.

Vous pouvez imaginer TeX comme une sorte d'organisation avec des « yeux », une « bouche », un « œsophage », un « estomac » et un « intestin ». Chaque partie de l'organisation transforme son entrée d'une certaine façon et passe l'entrée transformée à la prochaine étape. Les yeux transforment un fichier d'entrée en séquence de caractères. La bouche transforme la séquence de caractères en séquence de *tokens*, où chaque *token* est un caractère simple ou une séquence de contrôle. L'œsophage développe les *tokens* séquence de commandes primitives, qui sont également des *tokens*. L'estomac effectue les opérations indiquées par les commandes primitives, produisant une séquence de pages. Finalement, l'intestin transforme chaque page dans la forme requise pour le fichier PDF et l'y envoie.

La vraie composition est réalisée dans l'estomac. Les commandes demandent à TeX de composer tel caractère dans telle police, d'insérer de l'espace entre les mots, de finir un paragraphe et ainsi de suite. En commençant par des caractères composés individuellement et d'autres éléments typographiques simples, TeX construit une page comme un ensemble de boîtes de boîtes de boîtes. Chaque caractère composé occupe une boîte, et l'ensemble

fait une page entière. Une boîte peut contenir des boîtes plus petites mais aussi des ressorts et autres diverses choses. Les ressorts produisent de l'espace entre les cases les plus petites. Une propriété importante des ressorts est qu'ils peuvent s'étendre et se rétrécir ; ainsi TeX peut rendre une boîte plus grande ou plus petite en étirant ou en rétrécissant les ressorts. En général, une ligne est une boîte contenant une suite de boîtes de caractères, et une page est une boîte contenant une suite de boîtes de lignes. Il y a des ressorts entre les mots d'une ligne et entre les lignes d'une page. TeX étire ou rétrécit les ressorts sur chaque ligne afin de justifier une marge droite sur la page et les ressorts de chaque page afin de faire des marges inférieures des différentes pages égales. D'autres genres d'éléments typographiques peuvent également apparaître dans une ligne ou sur une page, mais nous ne les traiterons pas ici. En tant que processus d'assemblage de pages, TeX a besoin de couper des paragraphes en lignes et des lignes en pages. En effet, l'estomac voit d'abord un paragraphe comme une longue ligne. Il insère des coupures de ligne afin de transformer le paragraphe en séquence de lignes de la bonne longueur, exécutant une analyse plutôt raffinée afin de choisir l'ensemble de coupures qui semble la meilleure pour le paragraphe. L'estomac suit un processus semblable mais plus simple afin de transformer une séquence de lignes en page. Essentiellement l'estomac accumule des lignes jusqu'à ce que plus aucune ligne ne puisse rentrer dans la page. Il choisit alors un endroit simple pour couper la page, en mettant les lignes avant la coupure sur la page courante et en sauvant les lignes après la coupure pour la page suivante.

*TeX pour l'impatient par Paul W. Abrahams, Kathryn A. Hargreaves et Karl Berry,
traduction par Marc Chaudemanche sous licence GFDL.*

i

Pour plus d'informations à ce sujet, nous pouvons consulter un livre sur Tex. La référence, en anglais, est « *The TeXbook* » de Donald Knuth, mais nous pouvons également regarder « Apprendre à programmer en TeX » de Christian Tellechea ou « Tex pour l'impatient » en français.

5.2. Les groupes

Profitons de cette partie pour parler de quelques principes de TeX. Nous allons parler des groupes. Un groupe est une zone délimitée par des *tokens* spéciaux où les modifications, les assignations et les définitions sont locales (sauf si l'on indique qu'elles sont globales). Pour mieux comprendre comment ils agissent, considérons la commande `\em` qui permet de passer en « mode emphase ».

```
1 \begin{document}
2   On est dans le document. \em On a utilisé la commande pour
   mettre cette phrase en emphase.
3
4   On est dans un nouveau paragraphe.
5 \end{document}
```

I. Créer vos premiers documents

On remarque que **tout** ce qui suit `\em` est mis en emphase. Ça peut être un peu embêtant et pour limiter l'effet de la commande `\em` à une partie du document, nous allons la placer dans un groupe. La modification (le passage à l'emphase) sera alors locale et ne se fera que dans le groupe. Essayons alors le code qui suit.

```
1 \begin{document}
2   On est dans le document. {\em On a utilisé la commande pour
   mettre cette phrase en emphase.}
3
4   On est dans un nouveau paragraphe.
5 \end{document}
```

Ici, nous avons placé la phrase dans un groupe simple en l'entourant d'accolade. Les accolades sont donc les *tokens* pour les groupes simples. Nous pouvons aussi utiliser les *tokens* `\bgroup` et `\egroup` qui sont respectivement équivalentes à `{` et `}` à quelques détails près.

D'un autre côté, nous avons les groupes semi-simples, délimité par les *tokens* `\begingroup` et `\endgroup`. Nous ne verrons pas la différence entre les deux, mais il faut savoir que nous ne pouvons pas fermer un groupe semi-simple avec le *token* du groupe simple ni fermer un groupe simple avec le *token* d'un groupe semi-simple. Ainsi le code qui suit ne compile pas.

```
1 On ouvre un groupe simple {\begingroup et un groupe semi-simple}
   dans ce document. \endgroup
```

Ce code provoquera l'erreur « Extra }, or forgotten \endgroup ».

i

La commande `\em` est une commande dite « à bascule » (on parle également de **déclaration**), car son effet s'applique dans tout le document à partir de son utilisation (en fait, jusqu'à ce qu'on utilise une commande qui permet de supprimer son effet), le groupe permettant de limiter son effet à une zone. Ici, nous avons utilisé la commande `\em` à titre d'exemple. Cependant, son usage n'est pas recommandée et nous verrons dans le chapitre suivant par quoi la remplacer.

5.3. Les environnements

5.3.1. Définition d'un environnement

Pour continuer ce chapitre, parlons un peu des environnements.

?

Pour commencer, qu'est-ce qu'un environnement ?

I. Créer vos premiers documents

Un environnement est un groupe semi-simple. Dans ce groupe, une macro est exécutée au début du groupe et une autre macro est exécutée à sa fin. Puisque la localité existe dans les groupes, les environnements permettent de délimiter une partie du document dans laquelle d'autres règles (de mise en page par exemple) s'appliquent. Un environnement s'ouvre avec la commande `\begin{nom}` et se ferme avec la commande `\end{nom}`.



À chaque `\begin` doit correspondre un `\end`, ou nous obtiendrons une erreur. Il ne faut donc pas oublier de fermer chaque environnement que l'on ouvre. De plus, nous ne pouvons pas faire deux environnements se chevaucher. Ils peuvent être imbriqués l'un dans l'autre, mais ne peuvent pas se chevaucher.

Tout comme les commandes normales, les environnements peuvent avoir des options et des arguments. En fait, certains environnements ont même des variantes étoilées. Il ne faut donc pas nous étonner si un jour vous voyons ceci.

```
1 \begin{environnement*}[options]
2
3 \end{environnement*}
```

Dans la suite de ce chapitre, nous verrons plusieurs environnements qui permettent de donner une sémantique à notre texte.

5.3.2. Que font vraiment les commandes `\begin` et `\end` ?

Sans rentrer dans les détails, nous pouvons voir ce que font les commandes `\begin` et `\end`. En gros, la commande `\begin{nom}` fait ceci :

- elle vérifie que la commande `\nom` existe (si ce n'est pas le cas, on obtient une erreur) ;
- elle garde en mémoire `nom` ;
- elle ouvre un groupe semi-simple ;
- elle exécute la commande `\nom`.

La commande `\end{nom}` fait cela :

- elle exécute la commande `\endnom` si elle existe et ne fait rien sinon ;
- elle vérifie que le nom correspond au dernier nom gardé en mémoire par `\begin` et si ce n'est pas le cas, on obtient une erreur (c'est pour cela qu'on ne peut pas faire deux environnements se chevaucher) ;
- elle ferme le groupe.

En gros, la commande `\begin` place juste une commande dans un groupe semi-simple.

```
1 \begin{nom}
2   Dans un environnement.
```

I. Créer vos premiers documents

```
3 \end{nom}
```

Ce code n'est que la manière en LaTeX d'écrire le code TeX qui suit.

```
1 \begin{group} \nom  
2   Dans un environnement.  
3 \end{nom} \end{group}
```

Notons alors que nous pouvons utiliser n'importe quel nom de commande avec `\begin`. Par exemple, pour mettre une partie du texte en emphase, nous pouvons utiliser ce code.

```
1 \begin{em}  
2   Tout ce qui est là est en dans un groupe semi-simple dans lequel  
   on a utilisé la commande em.  
3 \end{em}
```

Ce chapitre était plutôt court, et pourtant nous y avons vu les environnements qui sont une notion importante en LaTeX. Notons bien qu'une compréhension fine des *tokens* et des *catcodes* n'est pas essentielle pour la suite.

6. De la sémantique

Le chapitre précédent commençait déjà le travail que nous allons faire ici. Nous allons nous construire un document qui a du sens et laisser LaTeX faire apparaître la signification des mots.

6.1. L’emphase

Pour commencer ce chapitre, nous allons faire un petit test. Compilons le code qui suit.

```
1 \documentclass[a4paper, 12pt, french]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \begin{document}
7   Ceci est \emph{important}.
8 \end{document}
```

?

Qui peut dire ce que fait la commande `\emph` ?

Un indice, le titre de cette partie devrait nous aider.

Celui qui répond que la commande `\emph` sert à mettre le texte en italique a tout faux. La commande `\emph` sert à mettre du texte en emphase. Elle sert à indiquer que le texte passé en paramètre à la commande est important. L’italique n’est qu’une conséquence de cela. LaTeX met ce qui est important en italique (car ce sont les conventions typographiques), mais il aurait tout autant pu le mettre en gras ou en violet. `\emph` n’est pas une commande de police c’est une commande **sémantique**. D’ailleurs, en écrivant `\emph{Dans cette phrase importante, \emph{ce mot} est vraiment très important}`, nous remarquons que « ce mot » est écrit en romain¹. L’emphase est donc faite dans un texte déjà important en passant à l’écriture romaine.

On aborde là un aspect fondamentale de la philosophie des *TeXniciens*. Nous devons faire la distinction entre le document et son rendu visuel. Notre rôle à nous est d’écrire un code avec la plus grande valeur sémantique possible. Ensuite, LaTeX se charge de mettre tout ceci en page. Non seulement cela assure une grande cohérence (tout ce qui est important sera écrit de la même manière, tout les titres de même niveau auront la même taille...), mais cela facilite aussi le passage au travail visuel.

LaTeX a des commandes de changement de police, de taille, de couleur... Mais, les utiliser comme ça ne rime à rien. Ces commandes sont judicieuses à utiliser lorsqu'on crée nos commandes. Prenons un exemple : nous écrivons un livre dans lequel nous voulons mettre tous les noms de lieu en petites capitales. Nous pourrions bien sûr, chaque fois que nous écrivons un lieu, utiliser une commande pour le mettre en petites capitales... Mais ce ne serait ni élégant, ni en conformité avec la philosophie de LaTeX. La bonne méthode serait de :

- créer une commande `\lieu` qui prendrait un argument obligatoire et le mettrait en petites capitales ;
- utiliser la commande `\lieu` à chaque fois qu'on écrit un lieu pour indiquer que c'est un lieu.

Il faut bien remarquer le changement de point de vue : on ne veut pas écrire qu'il faut le mettre en petites capitales, on veut indiquer qu'il s'agit d'un lieu. De plus, si plus tard on veut les lieux en bleus, il suffira de modifier la commande `\lieu`. Ainsi, si on veut que les mots importants soient en rouge gras, il faut modifier la commande `\emph`.

i

Nous verrons plus tard comment changer de polices, comment créer une nouvelle commande ou en modifier une existante.

En résumé, la commande `\emph` est la commande d'emphasis de LaTeX. Elle prend un argument obligatoire, et LaTeX s'occupe de mettre en forme cet argument pour que le lecteur comprenne qu'il s'agit de quelque chose d'important.

Dans le chapitre précédent, nous avons également vu la commande à bascule `\em` qui permet de passer en mode emphase. Contrairement à `\emph` qui est, rappelons-le, une commande courte, elle permet de mettre en emphase du contenu contenant la primitive `\par` (donc un changement de paragraphe). Cependant, `\emph` doit lui être préférée et donc nous n'utiliserons `\em` que très rarement voire jamais (il y a rarement besoin de mettre plusieurs paragraphes en emphase dans un texte).

6.2. Les listes

L'emphase c'est bien, mais avec ça seulement, on n'ira pas loin. On aimerait bien faire quelque chose d'autre histoire de changer. Les listes c'est bien, ça, non ?

Oui, les listes c'est bien, et ce qui est encore mieux, c'est que les listes sont très faciles à faire avec LaTeX. En plus, on peut faire trois types de listes. Cela nous fait trois nouveaux environnements à découvrir.

6.2.1. Les listes à puces

Les listes à puces correspondent à ceci :

- élément de la liste ;

1. L'écriture romaine ↗ correspond à l'écriture « normale », avec des caractères droits.

I. Créer vos premiers documents

- autre élément ;
- un dernier pour la route.

Pour les faire, il faut utiliser l'environnement `itemize`. À l'intérieur de celui-ci, il faut utiliser la commande `\item` pour une nouvelle entrée. Cette entrée peut tenir sur plusieurs paragraphes.

```
1 \begin{itemize}
2   \item Le premier item.
3   \item On fait un nouvel item.
4
5       On a sauté une ligne pour le faire sur plusieurs
        paragraphes.
6 \end{itemize}
```

i

En utilisant `babel` avec l'option `french`, les puces sont remplacées par des tirets.

6.2.2. Les listes numérotées

Les listes numérotées se construisent de la même manière que les listes à puces grâce à la commande `\item`. Seul l'environnement change : pour les listes numérotées, il faut utiliser l'environnement `enumerate`. La numérotation est faite automatiquement.

1. Premier.
2. Deuxième.
3. Troisième.

Cette liste est obtenue avec ce code.

```
1 \begin{enumerate}
2   \item Premier.
3   \item Deuxième.
4   \item Troisième.
5 \end{enumerate}
```

Là encore, les entrées peuvent être sur plusieurs paragraphes.

6.2.3. Les descriptions

Les listes de description s'utilisent comme les autres listes avec la commande `\item`. L'environnement à utiliser est cette fois `description`. Cet environnement permet de mettre en valeur une partie du texte (par défaut en gras). Pour cela, il faut passer en argument facultatif à la commande `\item` la partie du texte à mettre en valeur. Si nous ne passons aucun argument facultatif, aucun texte ne sera mis en valeur (et dans ce cas, nous devrions douter de l'utilité de notre liste). Voici un code d'exemple.

```
1 \begin{description}
2   \item[Zeste de Savoir] (ou \emph{ZdS} pour les intimes) est un
      site communautaire de partage libre de connaissances.
3   \item[Clem] est la mascotte officielle du \emph{ZdS}.
4   \item[\bsc{Karnaj}] est l'auteur de ce tutoriel.
5 \end{description}
```

6.2.4. De l'utilisation des listes

Ce chapitre traite de la sémantique, alors nous ne pouvons que dire qu'il faut utiliser les listes à bon escient. Si nous avons besoin de quelque chose avec une numérotation en lettres, nous pourrions utiliser une liste de description en mettant en valeur chaque lettre. Mais ce que nous voulons faire correspond plus à une liste numérotée qu'à une liste de description. Si nous voulons changer la numérotation, il faut changer le fonctionnement de l'environnement `enumerate` et non détourner l'environnement `description` de son usage.

Donc il faut utiliser les listes pour faire ce qu'elles sont censées faire :

- `itemize` pour une liste banale ;
- `enumerate` pour une énumération ;
- `description` pour une liste où quelque chose doit être mis en valeur (un dictionnaire par exemple).

6.3. Les citations et les vers

Nous allons maintenant voir trois environnements particuliers. Ils permettent de faire des citations hors-texte. Ces trois environnements indentent le texte pour le séparer du texte normal.

6.3.1. L'environnement `quote`

L'environnement `quote` est utilisé pour les citations courtes. Il s'utilise tout simplement en écrivant le texte à l'intérieur de l'environnement.

```
1 \begin{quote}
2   Notre première citation.
3 \end{quote}
```

Joli, non ? L'indentation fait clairement comprendre qu'il s'agit de quelque chose hors texte.



N'utilisez pas l'environnement `quote` pour des citations de plus d'un paragraphe.

Cette recommandation est due au fait que l'environnement `quote` ne place pas d'indentation au début des paragraphes, regardez :

```
1 \begin{quote}
2   Une citation sur plusieurs paragraphes. Le premier paragraphe de
3     la citation.
4   Le second paragraphe de la citation.
5 \end{quote}
```

Comme vous pouvez le voir, l'absence d'indentation laisse quasiment penser qu'il s'agit de deux citations différentes. En fait, il vaut mieux considérer chaque paragraphe dans un environnement `quote` comme une citation à part.



Cependant, si on veut faire deux citations, il vaut mieux utiliser deux environnements `quote`.

6.3.2. L'environnement `quotation`

L'environnement `quotation` est aussi là pour faire des citations.



Quoi ? Mais pourquoi avoir deux environnements de citations ?

L'environnement `quotation` est là pour pallier aux faiblesses de l'environnement `quote`. Il permet en effet de faire des citations sur plusieurs paragraphes car lui, il place l'alinéa de début de paragraphe. D'ailleurs, l'environnement `quotation` n'est utilisé **que** pour les citations de plus d'un paragraphe. En effet, dans le cas d'une citation à un paragraphe, on ne veut pas de l'indentation supplémentaire que donne `quotation`. On a donc :

- `quote` pour les citations de moins d'un paragraphe.
- `quotation` pour les citations longues, celles qui font plus d'un paragraphe.

I. Créer vos premiers documents

Compilez ce code et observez le résultat :

```
1 \begin{quotation}
2   Voici une citation longue.
3
4   Tellement longue qu'elle est sur plusieurs paragraphes.
5
6   Espérons que l'environnement saura distinguer les différents
   paragraphes.
7 \end{quotation}
```

6.3.3. L'environnement `verse`

L'environnement `verse` est un environnement qui vous permet d'écrire des vers (et donc des poèmes) facilement. Il n'y a que deux choses à savoir :

- On passe à un nouveau vers grâce à `\\`
- On passe à une nouvelle strophe en sautant une ligne (une strophe est donc un paragraphe)

L'environnement `verse` se charge de placer les vers bien et de laisser un espace entre chaque strophe. De plus, si un vers est trop long, il est coupé et la fin du vers est indenté. Ceci permet de ne pas la confondre avec le début d'un nouveau vers.

Testons donc cet environnement :

```
1 \begin{verse}
2 Et voilà je me lasse,\\
3 Tout cela me dépasse.\\
4 Le fil de mon ennui,\\
5 Tendue à l'infini,\\
6 Est rythmé par l'envie\\
7 D'une toute autre vie.\\
8 \end{verse}
```

Il s'agit de vers écrit par [Phigger](#) sur le [forum poétique des zesteurs](#). Si vous vous sentez l'âme poétique, je vous invite à aller poster là-bas (et à rédiger en LaTeX pour vous entraîner).

Avec ce chapitre fort en informations, nous avons déjà de quoi faire de bons documents et nous avons fait notre entrée dans le monde de la sémantique. Nous avons également vu les environnements, une notion qui est très importante. Les listes et l'emphase sont très utilisées. Les citations le sont un peu moins mais restent utiles.

Au programme du prochain chapitre, quelque chose de moins utilisé, mais de très puissant.

7. Annotations et références

C'est parti pour un nouveau chapitre. Nous allons voir les systèmes de références et d'annotations de LaTeX.

7.1. Notes

Il y a deux chapitres, nous avons parlé de la commande `\thanks` qui permettait d'obtenir une note de bas de page afin de donner des informations complémentaires à propos de l'auteur et du titre. Néanmoins, cette commande ne sert justement qu'à ça et l'utiliser en dehors des commandes `\author` et `\title` ne produira pas le résultat escompté. Nous allons maintenant voir comment faire pour obtenir des notes.

7.1.1. Notes de bas de pages

7.1.1.1. La commande `\footnote`

Pour écrire des notes de bas de page, nous pouvons utiliser la commande `\footnote`. Il suffit de la placer après le mot (ou le groupe de mots) à annoter en lui passant en paramètre la note en question. Cette note se retrouvera en bas de la page du texte annoté. LaTeX s'occupe de numéroter chacune des notes. Ainsi, le code...

```
1 \documentclass[a4paper, 12pt, french]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \begin{document}
7   Ceci est un document avec une note de bas de page
   ici\footnote{voici la note}.
8 \end{document}
```

... Permet d'obtenir le résultat suivant.

Ceci est un document avec une note de bas de page ici¹

Les notes de bas de page sont beaucoup plus utilisées que les notes de marges. Non seulement elles permettent d'écrire plus d'informations, mais en plus elles sont numérotées et donc sont plus précises que les notes de marge.

7.1.1.2. La commande `\footnotemark`

La commande `\footnote` répond à la plupart de nos besoins. Cependant, il peut arriver (c'est quand même assez rare) qu'elle ne soit pas satisfaisante. Ceci arrive majoritairement dans deux cas :

- on obtient une erreur avec `\footnote` ;
- on veut gérer soit même la numérotation (cas assez fantaisiste).

Nous pouvons alors utiliser la commande `\footnotemark`. Cette commande est à placer à côté du texte à annoter (comme `\footnote`), mais on ne lui passe pas le texte à annoter en argument. On lui passe en paramètre facultatif le numéro que l'on veut donner à la note. Puis, pour indiquer à LaTeX le texte associé à ce numéro, il faut utiliser la commande `\footnotetext`. Elle prend en paramètre le texte à annoter et en paramètre facultatif le numéro de la note. De cette manière, `\footnotemark` permet de choisir l'emplacement de la note, et `\footnotetext` son contenu. Ainsi, pour avoir la même note que précédemment il faut utiliser ce code.

```
1 \documentclass[a4paper, 12pt, french]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \begin{document}
7   Ceci est un document avec une note de bas de page
8     ici\footnotemark[1].
9   \footnotetext[1]{voici la note}
10 \end{document}
```



Cette méthode nécessite deux compilations. La première pour lister toutes les `\footnotemark` et la seconde pour leur associer les notes.

Un des avantages de `\footnotemark` est de pouvoir faire référence plusieurs fois à la même note de la manière suivante.

```
1 \begin{document}
2   Dans ce document, la même note de bas de page est
3     ici\footnotemark[1], mais aussi ici\footnotemark[1].
4   \footnotetext[1]{voici la note}
```

1. voici la note

```
4 \end{document}
```

7.1.2. Notes de marges

Les notes de marges sont les premières notes que nous verrons. Elles consistent à placer du texte dans la marge tout simplement. Pour les utiliser, il faut utiliser la commande `\marginpar`. Elle prend en paramètre le texte à afficher dans la marge.

Les notes de marges ne sont pas numérotées. Elles sont simplement placées dans la marge à côté de l'endroit où elles ont été appelées. Dès lors, il est conseillé de ne faire que des notes courtes (et de réserver les notes de marges aux documents dont les marges sont grandes), ceci afin de pouvoir bien identifier où la note a été appelé dans le document et de ne pas avoir un document trop chargé.



Il vaut mieux éviter d'utiliser la commande `\marginpar` dans une autre commande. Si dans une commande `\emph` elle se comporte comme d'habitude, elle ne peut pas être utilisée dans une commande de sectionnement par exemple et peut provoquer des comportements inattendus.

Voyons maintenant un exemple d'utilisation de cette commande.

```
1 \documentclass[a4paper, 12pt, french]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \begin{document}
7   Ceci est un document avec une note dans la marge
8     \marginpar{voici la note}.
9 \end{document}
```

7.2. Le système de références

LaTeX dispose d'un système de références très puissant. Il permet de faire des références croisées très facilement et donc de renvoyer à un autre passage du document sans trop de soucis. Il permet ainsi de faire référence à une section, à une image et à d'autres parties du document. En fait, il permet même de rappeler la page où se trouve l'objet référencé. Ceci est utile notamment dans le cas de documents à imprimer. Dans le cas de documents destinés à rester numérique, on préférera placer des liens hypertextes plutôt que de donner la page.

7.2.1. Un système d'étiquettes

Le système de références de LaTeX est un système à base d'étiquette c'est à dire à base de label. On pose une étiquette à un endroit du document pour y faire référence plus tard. Ceci permet – à condition de bien nommer ses étiquettes – de pouvoir faire référence à n'importe quelle partie du document plutôt facilement.

Pour placer une étiquette, il faut utiliser la commande `\label` à l'endroit où on veut placer cette étiquette. Elle prend en paramètre un nom qui permettra d'identifier cette étiquette. Voici un exemple.

```
1 \section{une première section}
2 \label{etiquette vers une section}
```

Bien sûr, les noms doivent quand même être assez explicite et celui que nous venons d'utiliser ne l'est pas du tout et ne sera donc jamais utilisé.

En fait, il est d'usage de construire les noms d'étiquette en faisant précéder le nom choisi par le type de l'élément auquel on fait référence. Ainsi si on fait référence à une section on aura `sec:nom` (ou `section:label`) et si on fait référence à un tableau ce sera plutôt `tab:nom`. Cette convention permet d'être plus clair et plus précis. De plus, elle limite le nombre de collisions possibles dans les choix du noms. On peut ainsi choisir un même nom pour un tableau et pour une section. Notre code précédent devient celui-là.

```
1 \section{une première section}
2 \label{sec:etiquette vers une section}
```

Ici, le `sec` permet de savoir qu'il s'agit d'une étiquette pour une section, ce qui est très utile lorsqu'on a des problèmes. Bien sûr, lorsqu'on change le plan de notre document, il ne faut pas oublier de changer le `sec` en `sous-sec` par exemple.

7.2.2. Faire nos références

Come nous l'avons déjà dit, le système de références de LaTeX nous permet d'accéder à deux données de l'étiquette :

- le numéro (tableau, section...) de l'objet étiqueté ;
- la page de l'étiquette.

Pour obtenir le numéro, il nous faut utiliser la commande `\ref`. Elle prend en paramètre le nom d'une étiquette. LaTeX s'occupe de gérer les numéros et nous n'avons donc rien d'autre à faire que d'utiliser cette commande. Pour obtenir la page de l'étiquette, c'est cette fois la commande `\pageref` qu'il faudra utiliser. Elle prend elle aussi le nom d'une étiquette en paramètre.



Pour que les références apparaissent sur le document, il faudra compiler deux fois. La première compilation permet de récolter tous les labels et les numéros qui leurs sont associés (ils sont écrits dans le fichier `.aux`) et la seconde permet d'écrire ces numéros sur le document.

Si nous ne compilons qu'une fois, les numéros ne seront pas présents et il y aura à la place des `??`. La même chose se produira si nous faisons une référence vers une étiquette qui n'existe pas. Si par contre aucun numéro n'apparaît, c'est qu'aucun numéro ne peut être associé à l'étiquette correspondante.

Un code serait donc de ce genre.

```
1 \documentclass[a4paper, 12pt, french]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \begin{document}
7   \section{Notes}
8   \label{sec:notes}
9   Pour faire des notes, nous pouvons utiliser plusieurs commandes.
10  L'une d'elle permet
11  d'avoir des notes de marges, les deux autres des notes de bas de
12  page.
13
14  \section{Les références}
15  Dans ce document, nous avons parlé des notes dans la section
16  \ref{sec:notes} qui est à
17  la page \pageref{sec:notes}.
18 \end{document}
```

Ce document ne ferait qu'une page et le résultat produit serait vraisemblablement celui-là.

```
1 Dans ce document, nous avons parlé des notes dans la section 1 qui
   est à la page 1.
```

Nous avons dit que la numérotation était gérée ; elle est vraiment gérée entièrement. Si par exemple nous faisons référence à une annexe, ce sera bien la numérotation alphabétique qui sera utilisé. En revanche, il nous faut noter que si on utilise une référence sans sectionnement (sans section, sans chapitre, etc.), les références ne seront pas numérotées et donc nous n'aurons pas de résultat.

7.3. Bien les utiliser

7.3.1. Conseils d'utilisations

Les systèmes d'annotations et de références doivent maintenant nous sembler simples à utiliser. Pourtant, ils sont pleins de petits pièges et des conseils d'utilisation ne seront pas de trop. Le meilleur conseil est à propos du nom des étiquettes qui se doit d'être clair.

Ce conseil peut sembler évident, mais non. Déjà, trouver un nom clair pour une étiquette n'est pas si facile.

?

Mais c'est quoi un nom clair au fait ?

C'est la question à laquelle il nous faut répondre.

Nous l'avons déjà dit, mais répétons le, un nom clair doit permettre de comprendre à quel type d'objet on fait référence. Nous le faisons en rajoutons les préfixes comme **sec**.

De plus, un nom clair est relativement **court**. En effet, les noms à rallonge sont rarement clairs et nous aurons sûrement des erreurs d'écriture du nom de l'étiquette. Cependant, il ne faut pas chercher à faire court à tout prix car un nom clair doit **avant tout** rappeler le sujet auquel on voudrait renvoyer. Par exemple, le nom **sec:un** n'est pas du tout clair sur ce point et pourtant il est très court. On comprend qu'il fait référence à la première section mais on ne sait pas du tout quel est le sujet de la première section en question. Et si on rajoutait une nouvelle section avant la première ? Il nous faudrait alors réécrire toutes nos étiquettes de section de ce type. Préférons alors une étiquette du genre **sec:references** si la section en question traite des références.

7.3.2. Ce n'est pas...

Ce système est très puissant et on pourrait penser à l'utiliser pour faire certaines choses qui ne sont en fait pas des références. N'oublions pas que LaTeX met un point sur la sémantique, alors utiliser des références lorsqu'il n'a pas lieu d'en utiliser, c'est faire (disons le franchement) du n'importe quoi. Donc voyons ce que n'est pas.

7.3.2.1. Un système d'index

Donc tout d'abord ce n'est pas un système d'index. Non, ce n'en est vraiment pas un ! Créer un index avec des références, c'est se fatiguer pour pas grand chose. Non seulement cela demanderait de créer une étiquette à chaque endroit du document où le mot indexé est écrit, mais en plus ces étiquettes ne devraient pas avoir le même nom ce qui complique énormément les choses. Dans un document d'une petite dizaine de pages, cela n'est pas gênant, mais imaginons le mal que cela donnerait dans un document d'une cinquantaine de page.

7.3.2.2. Un système de bibliographie

Et ce n'est pas non plus un système de bibliographie. Nous pouvons toujours essayer de l'utiliser pour cela, mais ce serait fatigant, pas beau et pas ergonomique. Là encore, il faut penser aux grands documents pour voir le désavantage de cette méthode.



Mais dans ce cas, comment on fait les index et les bibliographies ?

Il ne faut pas s'inquiéter pour cela. Nous verrons plus tard que si LaTeX a pour but de nous aider à écrire nos documents, d'autres logiciels ont pour but de nous aider à faire d'autres choses, notamment les bibliographies et les index. Ils s'intègrent parfaitement à LaTeX et permettent d'obtenir facilement le résultat attendu. C'est là encore l'un des points-forts de la philosophie des *Texniciens* : un outil pour faire une chose, mais pour la faire bien.

C'est la fin d'un nouveau chapitre. Celui-ci était plus court que le précédent et nous a permis de voir le système simple et flexible utilisé par LaTeX pour annoter un document.

Mais la vraie morale (et elle est à retenir) c'est qu'avec LaTeX on n'utilise pas n'importe quoi pour faire quelque chose, on utilise l'outil adapté. Couper sa viande avec une cuillère, c'est possible, mais utiliser un couteau c'est nettement mieux. Autant laisser la cuillère pour de la glace par exemple (où le couteau est cette fois inadapté). Cela ne marche pas que dans la vie courante. En informatique aussi, c'est mieux d'utiliser les outils adaptés.

8. Différents types de documents

Le dernier chapitre de cette section nous apprendra à créer d'autres types de documents. Nous apprendrons également à organiser nos sources LaTeX. Pour cela, nous verrons qu'il est possible d'écrire dans plusieurs fichiers plutôt que dans un seul.

8.1. Des classes pour de plus gros documents

En plus de la classe `article`, LaTeX nous propose deux classes pour les documents « classiques ».

8.1.1. La classe `book`

Pour commencer, la classe `book`, comme son nom l'indique, permet d'écrire des livres. Cette classe rajoute la commande `\chapter` comme commande de sectionnement, les chapitres étant le niveau de sectionnement juste au dessus des sections et `\chapter` a donc le poids 0. Les chapitres commencent sur une nouvelle page.

Les options de la classe `article` sont toujours valides pour la classe `book`. Notons cependant que par défaut c'est l'option `titlepage` qui est présente pour la classe `book`.

De plus, nous pouvons demander que les nouveaux chapitres commencent toujours sur une page impaire (c'est ce qui est fait dans beaucoup de livres ; les chapitres commencent sur la page de droite). Pour cela, nous utiliserons l'option `openright`, l'option par défaut étant `openany` qui permet aux chapitres de commencer sur n'importe quelle page.

```
1 \documentclass[a4paper, 12pt, openright, french]{book}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \begin{document}
7   \chapter{Le premier}
8     \section{Une section}
9   \chapter{Un autre}
10     \section{Une section}
11       \subsection{Une sous-section}
12 \end{document}
```

8.1.1.1. Structure du document

De plus, la classe `book` vient avec trois commandes (en plus de la commande `\appendix`) pour structurer encore plus notre document en lui donnant un prologue, un corps et un épilogue. Ces commandes modifient le type de numérotation des pages et des chapitres de la manière suivante.

Commande	Utilisation	Numérotation des pages	Numérotation des chapitres
<code>\frontmatter</code>	Prologue	Chiffres romains minuscules	Non présente
<code>\mainmatter</code>	Corps	Chiffres arabes	Chiffres arabes
<code>\appendix</code>	Annexe	Chiffres arabes	Lettres majuscules
<code>\backmatter</code>	Épilogue	Chiffres arabes	Non présente

TABLE 8.2. – Numérotation des différentes parties d’un document de la classe `book`.
Un exemple aidera à s’en faire une meilleure idée.

```

1 \documentclass[a4paper, 12pt, french]{book}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \title{Exemple de livre}
7 \author{ZdS}
8
9 \begin{document}
10     \maketitle
11     \tableofcontents
12     \frontmatter
13     \chapter{Prologue}
14
15     \mainmatter
16     \chapter{Chapitre}
17         \section{Section}
18
19     \appendix
20     \chapter{Annexe}
21         \section{Section d'annexe}
22
23     \backmatter
24     \chapter{Épilogue}
25 \end{document}

```

On remarque de plus que la numérotation recommence de zéro quand on utilise la commande `\mainmatter`.

8.1.2. La classe `report`

Finalement, pour rédiger un rapport, nous pouvons utiliser la classe `report`. Cette commande rajoute une autre commande de sectionnement (encore une), la commande `\part`, qui est un niveau au dessus des chapitres (et a donc un poids de -1).

La classe `report` a les mêmes options que la classe `article`. En particulier, il n'y a pas d'option `openright` pour avoir les débuts de chapitres sur des pages impaires. La différence majeure est que l'option par défaut pour le titre est `titlepage`.

De plus nous n'avons pas accès aux commandes `\frontmatter`, `\mainmatter` et `\backmatter` avec la classe `report`.

Finalement, la classe `report` ressemble beaucoup à une classe `article` avec deux commandes de sectionnement supplémentaires (`\chapter` et `\part`).

```
1 \documentclass[a4paper, 12pt, french]{report}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \title{Exemple de rapport}
7 \author{ZdS}
8
9 \begin{document}
10   \maketitle
11   \tableofcontents
12   \part{Partie}
13     \chapter{Chapitre}
14       \section{Section}
15   \part{Autre partie}
16     \chapter{Autre chapitre}
17       \section{Autre section}
18 \end{document}
```

Notons que la numérotation des chapitres ne repart pas de zéro quand on change de partie. Ce comportement peut être changé, nous verrons cela dans la partie suivante.

8.2. D'autres types de documents

En plus d'écrire des documents « textuels » dirons-nous, LaTeX nous permet d'écrire pleins d'autres types de documents. Nous allons donc voir d'autres types de documents qui sont offerts par les classes standards de LaTeX.

8.2.1. Des lettres

Pour ceux qui veulent écrire des lettres, LaTeX peut également le faire et propose une classe standard, `letter`, qui permettra de placer les éléments comme l'adresse ou encore la signature de l'expéditeur. Mais cette classe place les éléments en suivant les conventions anglaises. Pour suivre les conventions françaises, nous allons plutôt utiliser la classe `lettre`.

L'élément principal de cette classe est l'environnement `letter` dans lequel nous écrivons notre lettre. Nous pouvons ainsi écrire plusieurs lettres dans un seul document en utilisant plusieurs

I. Créer vos premiers documents

fois cet environnement. Il prend en paramètre le nom du destinataire de la lettre. À l'intérieur de cet environnement, on écrit notre lettre en trois parties principales.

- Les salutations, avec la commande `\opening` qui prend en paramètre... Les salutations.
- Le corps de la lettre.
- La formule de politesse avec la commande `\closing` qui prend un paramètre.

On a alors quelque chose du genre. Remarquons que LaTeX ajoute la date du jour automatiquement.

```
1 \documentclass[a4paper, 12pt, french]{letter}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \begin{document}
7 \begin{letter}{Destinaire 1}
8   \opening{Cher}
9   Une première lettre
10  \closing{Meilleurs voeux,}
11 \end{letter}
12
13 \begin{letter}{Destinataire 2}
14   \opening{Madame, monsieur}
15   Une première lettre
16   \closing{Respectueusement,}
17 \end{letter}
18 \end{document}
```

Les salutations et la formule de politesse ne sont bien sûr pas obligatoires. En plus de ces commandes, nous avons les commandes suivantes.

- `\name` pour préciser le nom de l'expéditeur.
- `\address` pour son adresse.
- `\telephone` pour son numéro de téléphone.
- `\date` pour préciser la date.
- `\ps` pour rajouter des *post-scriptum* (elle prend deux paramètres, le label à afficher, et le texte à afficher).
- `\cc` pour indiquer les personnes en copie de la lettre.
- `\encl` pour préciser les pièces jointes à la lettre

Nous avons donc de quoi écrire une vraie lettre plutôt facilement. Et il y en a quelques autres de plus. Les commandes `\name`, `\address`, `\date` et `\telephone` sont généralement utilisées dans le préambule, `\ps`, `\cc` et `\encl` là où on veut les voir apparaître.

```
1 \documentclass[a4paper, 12pt, french]{lettre}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{babel}
5
6 \name{Karnaj}
7 \address{Tout}
```

```
8 \date{\today}
9
10 \begin{document}
11 \begin{letter}{Destinataires}
12   \opening{À tous,}
13     Bonne année
14   \closing{Meilleurs voeux,}
15   \ps{PS :}{que la force soit avec vous}
16   \ps{PS 2:}{Geronimo}
17   \cc{Tout le monde}
18   \enccl{Des chocolats}
19 \end{letter}
20 \end{document}
```

Ce [tutoriel](#) nous permettra d'en apprendre plus sur la classe `lettre`.

8.2.2. Des présentations

LaTeX permet aussi de faire des présentations avec la classe standard `slides`. Néanmoins, la classe `slides` n'offre pas beaucoup de possibilités simples et est donc beaucoup moins utilisée que des classes plus complètes. C'est pourquoi nous n'allons pas la voir ici.

Pour faire des présentations, nous pouvons regarder la classe `beamer` qui est très bien documentée ; de nombreux tutoriels à son sujet sont trouvables sur Internet. Voici un exemple de ce qu'elle permet de faire (ici quatre *slides* faites avec le thème Metropolis).

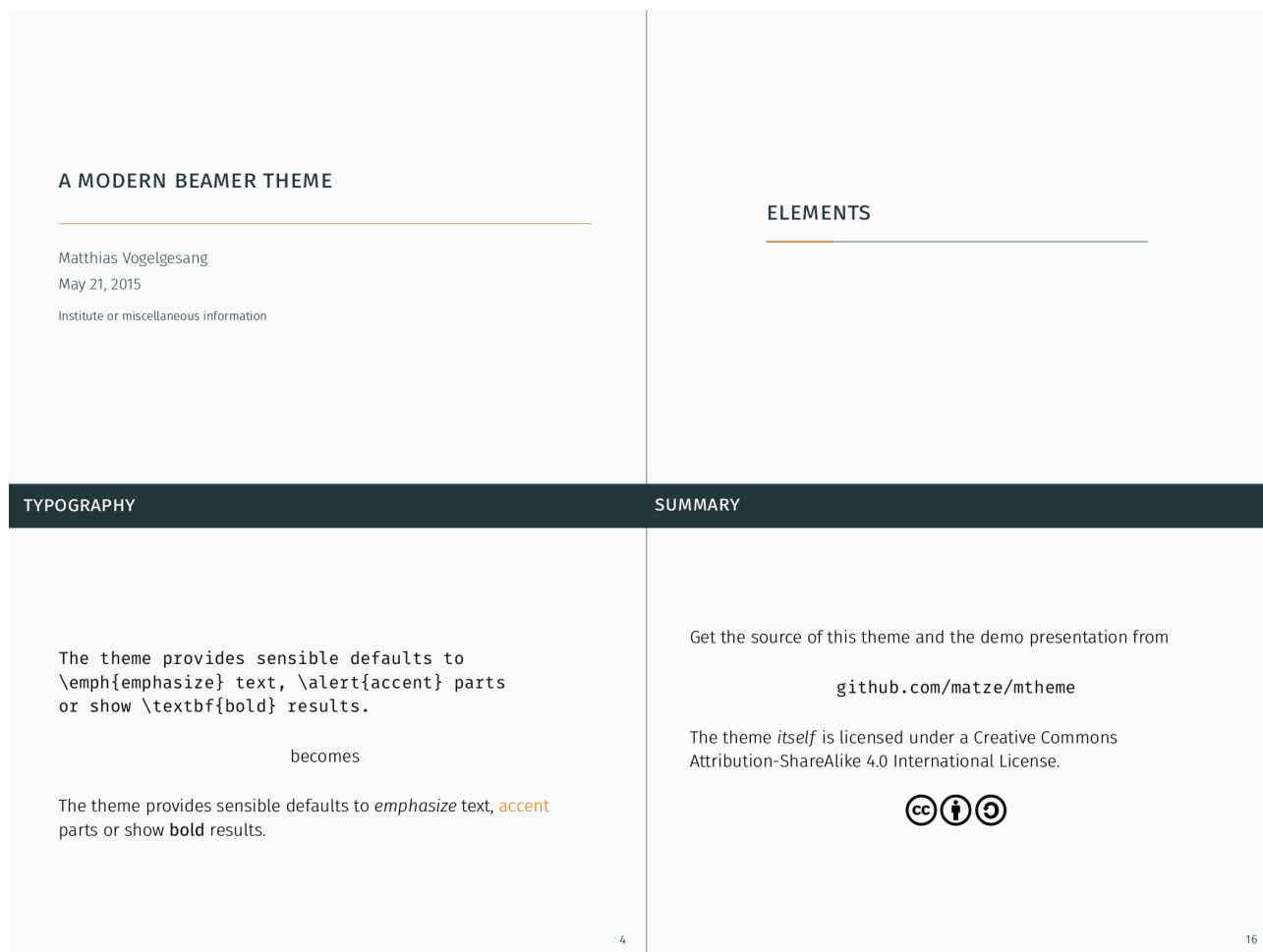


FIGURE 8.1. – Exemples de *slides* (dépôt [Github de Metropolis](#)).

Sur le dépôt, nous trouvons également [un exemple de présentation](#) . Nous pouvons déjà, si nous le souhaitons, regarder comment faire des présentations ; nous avons désormais les compétences pour cela. Néanmoins, certains des chapitres suivants, par exemple sur l’inclusion de tableaux et d’images pourront être utiles avant cela.

Et ce n’est pas tout, il est également possible de faire d’autres types de document avec LaTeX ! Des affiches avec `baposter` ou encore `modernposter`, des CVs avec des classes comme `moderncv`, etc. En fait il y a des classes pour faire à peu près tout (mais pas le café).

8.3. Organiser ses fichiers

Lorsque nous écrivons un grand document, ce n’est pas très agréable d’avoir tout le code dans un même document et on peut se perdre dans le fichier source très facilement. Nous avons la possibilité d’inclure des fichiers dans d’autres, et c’est cette possibilité que nous allons utiliser.

8.3.1. Structurer notre projet

Notre but est alors d’écrire dans plusieurs fichiers, voire même dans plusieurs dossiers. Cela permettrait d’avoir un fichier par section (ou même par sous-section), et donc d’être beaucoup mieux organisé. Nous aurions alors un fichier général qui inclut les autres fichiers. Par exemple, nous pourrions avoir une arborescence de ce type.


```
1 conclusion.tex
2 introduction.tex
3 projet.tex
4 section_1/
5     section1.tex
6     sous_section1.tex
7     sous_section2.tex
8 section_2/
9     section2.tex
10    sous_section1.tex
11    sous_section2.tex
12    sous_section3.tex
```

Cela permet alors d'avoir le sectionnement représenté dans l'arborescence de notre projet. Le fichier `projet.tex` est le fichier principal du projet. C'est dans ce fichier que nous aurons notre préambule, notre `\begin{document}` et notre `\end{document}`. Il inclut l'introduction, la conclusion et les fichiers `section1.tex` et `section2.tex`. Les fichiers `section1.tex` et `section2.tex`, eux, incluent les fichiers associés à leurs sous-sections respectives.

8.3.2. La commandes `\input`

Pour inclure un fichier dans un autre, nous pouvons utiliser la commande `\input` qui prend en paramètre le chemin du fichier à inclure. En gros, elle va prendre le contenu de ce fichier à la place du `\input`. Par exemple, on peut avoir un fichier `section.tex` et le fichier principal `projet.tex`.

```
1 % Le fichier section.tex
2 \section{Une section}
```

```
1 % le fichier projet.tex
2 %préambule
3 \begin{document}
4     \input{section}
5 \end{document}
```

Ici, nous n'avons pas mis l'extension `.tex` du fichier, elle n'est pas obligatoire puisque LaTeX cherche par défaut un fichier avec l'extension `.tex`. Néanmoins, si nous voulons inclure un fichier avec une autre extension (c'est possible), il faudra préciser cette dernière.

La commande `\input` fait littéralement un copier-coller ; elle ne compile pas le fichier pour ensuite prendre son résultat. Cela signifie par exemple que le code qui suit ne marche pas (on garde le même fichier `section.tex` que plus haut).

```
1 \begin{document}
2     \section{\input{section}}
```

```
3 \end{document}
```

En effet, on ne peut pas déclarer une section dans une section.

8.3.3. La commande `\include`

À côté de la commande `\input` se trouve la commande `\include` qui prend elle aussi le nom d'un fichier (notons qu'avec cette commande, il ne faut pas mettre l'extension du fichier). Elle permet elle aussi d'inclure le contenu du fichier dans, mais a néanmoins quelques différences avec la commande `\input`.

Premièrement, et c'est la différence la plus visible, la commande `\include` commence une nouvelle page avant d'inclure le fichier et en commence une autre après avoir inclus le fichier. Par exemple, elle est idéale pour les chapitre de livres, alors que pour des sections on préférerait sûrement utiliser `\input`.

De plus, un document qui est inclus avec la commande `\include` ne peut pas utiliser lui même une commande `\include`, ceci mènera à une erreur. On ne peut pas imbriquer les `\include`.

```
1 %fichier chap.tex
2 \chapter{Chapitre}
3 \include{section}
```

```
1 %fichier projet.tex
2 \maketitle
3 \include{chapitre}
```

Avec ce code nous obtenons l'erreur `\include cannot be nested`.

Au contraire, nous pouvons imbriquer des `\input` comme nous le voulons. En remplaçant les `\include` du code précédent par des `\input`, nous n'avons plus d'erreur.

?

La commande `\include` semble quand même plutôt limitée comparée à `\input`, pourquoi utiliser `\include` ?

Comme d'habitude il nous faut utiliser la commande adaptée à ce que l'on veut faire. Nous avons par exemple dit plus haut que `\include` pouvait être plus adaptée à l'inclusion de chapitres.

8.3.3.1. La commande `\includeonly`

La commande `\include` a un avantage qui est la commande `\includeonly`. Cette commande, qui ne peut être utilisée que dans le préambule, permet d'indiquer à LaTeX les fichiers à inclure. Elle prend en paramètre les noms des fichiers, séparés par une virgule.

```
1 \includeonly{chapitre_3, chapitre_4}
2
3 \begin{document}
```

```
4 \include{chapitre_1}
5 \include{chapitre_2}
6 \include{chapitre_3}
7 \include{chapitre_4}
8 \end{document}
```

Ici, seuls les fichiers `chapitre_3.tex` et `chapitre_4.tex` seront inclus. Cela permet d'accélérer la compilation et est donc utile si nous avons de très gros documents. Imaginons par exemple un livre de quatre chapitres. Une fois que les chapitres un et deux sont rédigés et ne seront plus changés, ce n'est plus la peine de les compiler pendant qu'on travaille sur les chapitres trois et quatre ; la commande `\includeonly` permet d'éviter de le faire.

8.3.4. Le package `subfiles`

Des *packages* comme `subfiles` que nous allons voir ici permettent de séparer notre code dans plusieurs fichiers et proposent des fonctionnalités supplémentaires. Avec `subfiles`, nous allons être capable de compiler chaque fichier indépendamment, chaque sous-fichier utilisant le préambule du fichier principal.

?

À quoi ça peut servir ?

Si nous écrivons un gros document de plusieurs chapitres, cela nous permet – si chaque chapitre est dans un document – d'obtenir un PDF pour le document complet, mais aussi des PDFs pour chaque chapitre.

L'utilisation de `subfiles` n'est pas très compliquée. Nous chargeons le package dans le fichier principal, et pour inclure les sous-fichiers nous n'allons pas utiliser `\input` ou `\include`, mais `\subfile`.

La grande différence se situe dans le contenu des sous-fichiers. Ceux-ci deviennent des documents LaTeX « complets » dans le sens où ils commencent par un `\documentclass` et ont un environnement `document`. En fait, ils sont de la forme suivante.

```
1 \documentclass[principal.tex]{subfiles}
2 \begin{document}
3   % Contenu
4 \end{document}
```

Les sous-documents peuvent alors être compilés seuls ; ils utiliseront les *packages* et le préambule du document principal (dans notre exemple, `principal.tex`). Finalement, on peut avoir cet exemple.

```
1 % Fichier principal.tex
2 \documentclass[a4paper, 12pt, french]{article}
3 \usepackage[utf8]{inputenc}
4 \usepackage[T1]{fontenc}
5 \usepackage{subfiles}
6 \usepackage{babel}
```

I. Créer vos premiers documents

```
7
8 \begin{document}
9   \subfile{chapitre_1}
10  \subfile{chapitre_2}
11 \end{document}
```

```
1 % Fichier chapitre_1.tex
2 \documentclass[principal.tex]{subfiles}
3 \begin{document}
4   \chapter{Le premier chapitre}
5     Son contenu.
6 \end{document}
```

```
1 % Fichier chapitre2.tex
2 \documentclass[principal.tex]{subfiles}
3 \begin{document}
4   \chapter{Le second chapitre}
5     Son contenu.
6 \end{document}
```

Grâce à ce chapitre, nous avons quelques cordes supplémentaires à notre arc de TeXnicien. Nous pouvons maintenant mieux gérer nos projets et nous avons une plus grande diversité de projets possibles.

Maintenant, vous connaissez les bases de l'utilisation de LaTeX. Mais que diriez-vous d'approfondir vos connaissances ? D'apprendre à intégrer des formules mathématiques, des tableaux et des images ? Et même de créer vos propres commandes ? C'est ce que nous verrons dans les prochains chapitres (et bien plus encore).

Deuxième partie

Compléter vos documents

II. Compléter vos documents

Nous savons maintenant comment produire des documents en LaTeX. Cependant, nos documents sont pour le moment plutôt basique et il nous faut donc apprendre à les complexifier, à ajouter d'autres données. Nous verrons comment faire cela dans cette partie. Au programme les maths, les images, les tableaux et bien d'autres choses.

9. Des Mathématiques

L'écriture des mathématiques est une des fonctions clés de LaTeX. En effet, nous avons accès à un système complet et efficace et disposons également de plusieurs *packages* pour nous faciliter l'écriture des formules. Dans ce chapitre, nous allons voir les bases de l'écriture de formules.

9.1. Écriture de base

9.1.1. Les modes mathématiques

Il nous faut distinguer deux manières d'écrire des mathématiques.

- La première consiste à écrire les mathématiques dans le message (on parle de mode **en ligne**) de la manière suivante : $f(x) = 2x$.
- L'autre mode (dit **hors texte**) met les formules mathématiques en évidence en dehors du texte. Par défaut, les formules écrites sont centrées. On a alors

$$f(x) = 2x.$$

Pour écrire des formules mathématiques en mode en ligne, il nous suffit de les délimiter par le symbole `$`. Nous pouvons également les écrire entre `\(` et `\)` ou dans l'environnement `math`.

```
1 Posons $f(x) = 2x$.
2
3 Posons \( f(x) = 2x \).
4
5 Posons \begin{math} f(x) = 2x \end{math}.
```

Pour des raisons de concision et de facilité de lecture, nous préférons utiliser le symbole `$`. C'est lui qui est usuellement utilisé.

Les mathématiques hors texte, elles, se placent entre `\[` et `\]` ou dans l'environnement `displaymath`, les deux écritures étant équivalentes. Là encore, pour des raisons de concision, nous allons préférer utiliser la première écriture.

```
1 Posons
2 \[
3   f(x) = 2x.
4 \]
5
6 Posons
7 \begin{displaymath}
8   f(x) = 2x.
9 \end{displaymath}
```

II. Compléter vos documents

Les mathématiques hors texte sont centrées par défaut. Néanmoins, il est possible d'aligner les mathématiques hors texte en passant l'option `fleqn` à `\documentclass`.



Pour écrire des mathématiques hors texte, il existe également la primitive de Tex `$$` (on écrira alors `$$f(x) = 2x$$`). Cependant, nous ne l'utiliserons pas, car, dans certains cas, elle peut altérer les espaces verticales placées avant la formule.

9.1.2. Syntaxe de base

Le mode mathématique doit être utilisé dès que l'on a à écrire des mathématiques. Remarquons par exemple la différence entre « f », en mode mathématique, et « `f` », en mode texte. L'écriture n'est pas la même. Par exemple, l'utilisation de lettres accentuées ne fonctionne pas et elles n'apparaîtront pas dans notre document.

D'autres choses diffèrent entre le mode mathématique et le mode texte. La gestion des espaces notamment est différente. En effet, l'espacement entre les symboles est géré automatiquement (suivant la nature du symbole) et le mode mathématique ne prend pas en compte les espaces du code source.

```
1 \[
2   Nous sommes en mode mathématique.
3 \]
```

Le résultat obtenu avec ce code n'est pas celui attendu. Notons que cela signifie que nous pouvons aérer notre code comme on le veut, `$x + 2$`, `$x+2$` et `$ x+ 2 $` étant équivalents. De plus, il est interdit de faire deux retours à la ligne consécutifs (c'est-à-dire d'utiliser la primitive `\par`) en mode mathématique sous peine d'obtenir une erreur.

9.1.3. Les commandes usuelles

Maintenant que nous connaissons la syntaxe de base et les erreurs à ne pas commettre, voyons les commandes usuelles du mode mathématique.



Les commandes que nous allons voir ici ne sont utilisables qu'en mode mathématique. Les utiliser en mode texte nous fera obtenir l'erreur explicite « missing \$ inserted ». Notons de plus que pour écrire le symbole `\`, nous ne devons pas utiliser la commande `\textbackslash` qui, comme son nom l'indique, est faite pour le mode texte, mais la commande `\backslash`.

9.1.3.1. Indice et exposants

Pour produire des indices et des exposants, il nous suffit d'utiliser les symboles `_` et `^`.

```
1 On produit un exposant et un indice :  $2^2$  et  $2_2$ .
```


II. Compléter vos documents

Si nous voulons placer un ensemble d'éléments en indice, il nous faudra le placer entre accolades ; 2^{23} et $2^{\{23\}}$ donnent alors deux résultats différents.

9.1.3.2. Fractions

Pour produire des fractions, il nous faut utiliser la commande `\frac`. Elle prend deux arguments, le numérateur et le dénominateur de la fraction. Par exemple, `\frac{1}{2}` donne $\frac{1}{2}$. Notons que la commande `\frac` ne produit pas le même résultat en mode en ligne qu'en mode hors texte. En mode hors texte, la fraction obtenue est plus grande. Pour obtenir le même résultat en mode en ligne, il nous suffit d'utiliser la commande `dfrac` (pour *display fraction*) à la place de la commande `frac`.

9.1.3.3. Racines

Les racines, elles, se composent en utilisant la commande `sqrt` qui prend en argument l'expression à placer sous la racine. Ainsi, `\sqrt{2}` donne $\sqrt{2}$. De plus, la commande `\sqrt` prend en paramètre facultatif l'ordre de la racine (qui est donc 2 par défaut). Ainsi, avec `\sqrt[3]{2}`, nous obtenons $\sqrt[3]{2}$.

9.1.3.4. Fonctions standards

Nous pouvons ressentir le besoin d'utiliser une fonction comme la fonction cosinus dans nos formules. Cependant, `\cos(x)` ne donne pas le résultat attendu, c'est-à-dire qu'on obtient $\cos(x)$ là où on voudrait plutôt $\cos(x)$. Heureusement, nous avons accès à un certain nombre de commandes pour la majorité des fonctions mathématiques usuelles. Elles ne prennent pas d'argument et se contentent d'écrire la fonction en romain et avec les bons espacements. Par exemple, le résultat précédent a été obtenu avec `\cos(x)`. Voici un tableau des fonctions disponibles.

Résultat obtenu	Commande
cos, cosh, arccos	<code>\cos</code> , <code>\cosh</code> , <code>\arccos</code>
sin, sinh, arcsin	<code>\sin</code> , <code>\sinh</code> , <code>\arcsin</code>
tan, tanh, arctan	<code>\tan</code> , <code>\tanh</code> , <code>\arctan</code>
cot, coth, csc	<code>\cot</code> , <code>\coth</code> , <code>\csc</code>
sec, arg, exp	<code>\sec</code> , <code>\arg</code> , <code>\exp</code>
ln, log, lg	<code>\ln</code> , <code>\log</code> , <code>\lg</code>
dim, deg, ker	<code>\dim</code> , <code>\deg</code> , <code>\ker</code>
det, gcd, hom	<code>\det</code> , <code>\gcd</code> , <code>\hom</code>
max, min, Pr	<code>\max</code> , <code>\min</code> , <code>\Pr</code>
inf, sup	<code>\inf</code> , <code>\sup</code>

9.1.3.5. Intégrales, sommes et limites

Nous disposons également de commandes pour ces éléments. La commande `\sum` donne le signe somme, `\int` le signe intégrale, `\prod` le signe produit et `\lim` le symbole limite. Ainsi, `\sum k` donne $\sum k$. Nous avons également accès à la commande `\oint` pour les intégrales sur des courbes fermées (le symbole \oint).

Pour placer les bornes inférieure et supérieure, la syntaxe reste simple puisqu'il nous suffit de les placer en indice et en exposant en utilisant `^` et `_`. On peut alors écrire ce code.

```
1 \[
2   \sum_{k = 0}^n k = \frac{n(n + 1)}{2}.
3 \]
```

Qui nous permet d'obtenir ce résultat.

$$\sum_{k=0}^n k = \frac{n(n+1)}{2}.$$

i

Notons que de même que pour la commande `\frac`, les résultats obtenus avec cette commande seront différents en mode en ligne, le code précédent donnant $\sum_{k=0}^n k = \frac{n(n+1)}{2}$.

9.1.3.5.1. Pour les limites En mode mathématique, nous pouvons avoir un « l » correspondant à la limite avec la commande `\ell` qui nous donne ℓ plutôt que l . Le symbole de l'infini est lui obtenu avec la commande `\infty`.

De plus, le symbole « tend vers » est obtenue avec la commande `\to`. On peut alors écrire

$$\lim_{x \rightarrow \infty} f(x) = \ell$$

avec le code qui suit.

```
1 \[
2   \lim_{x \to \infty} f(x) = \ell
3 \]
```

Nous avons également les commandes `\liminf` et `\limsup` qui fonctionnent de la même manière que `\lim` et donnent \liminf et \limsup .

9.1.3.6. Accents, vecteurs, conjugués, etc.

Nous avons dit qu'écrire une lettre accentuée dans notre code source n'avait aucun effet. Pour obtenir des accents, il nous faut utiliser des commandes.

Résultat obtenu	Commande
\acute{a} , \grave{a} , \hat{a} , \ddot{a} , \tilde{a}	<code>\acute{a}</code> , <code>\grave{a}</code> , <code>\hat{a}</code> , <code>\ddot{a}</code> , <code>\tilde{a}</code>
\check{a} , \breve{a} , \bar{a} , \dot{a} , \vec{a}	<code>\check{a}</code> , <code>\breve{a}</code> , <code>\bar{a}</code> , <code>\dot{a}</code> , <code>\vec{a}</code>

II. Compléter vos documents

La commande `\vec`, nous permet alors d'écrire des vecteurs en utilisant une flèche, et `\bar` nous permet d'écrire le conjugué d'un nombre complexe. Cependant, la longueur de l'accent ne s'adapte pas à la longueur du contenu, `\vec{ABC}` donnant \overrightarrow{ABC} . Pour régler ce problème, nous allons plutôt utiliser les commandes `\overrightarrow` (pour les vecteurs) et `\overline` (pour le conjugué) qui nous donnent la flèche et la barre de la bonne longueur. On obtient alors

$$\overrightarrow{ABC}, \overline{1+z}$$

avec le code qui suit.

```
1 \[ \overrightarrow{ABC}, \overline{1 + z} \]
```

i

Pour garder un document avec une bonne sémantique, il vaut mieux créer des commandes `\vecteur` et `\conjugue` qui appliquent respectivement `\overrightarrow` et `\overline` à leur argument.

Notons également les commandes `\widehat` et `\widetilde` qui sont les pendants de `\hat` et de `\tilde` aux longueurs adaptées et la commande `\underline` qui permet de souligner.

Avec les commandes que nous avons vu ici, nous avons déjà de quoi écrire une grande majorité de formules scientifiques. Et pour écrire plus facilement, certains *packages* nous donnent plusieurs commandes. Par exemple, le *package* **amsmath** nous fournit la commande `\text` qui nous permet d'écrire du texte en mode mathématique ou encore la commande `\binom` pour obtenir un coefficient binomial.

$\binom{n}{k}$ est le nombre de parties à k éléments d'un ensemble à n éléments

est alors obtenu avec ce code (notons également la commande `\dbinom` qui est à `\binom` ce que `\dfrac` est à `\frac`).

```
1 \[
2   \binom{n}{k} \text{ est le nombre de parties à } \$k\$ \text{ éléments d'un}
   \text{ ensemble à } \$n\$ \text{ éléments}
3 \]
```

Le *package* **amsmath** est quasiment incontournable lorsqu'on veut écrire des formules un peu complexes. Par exemple, les doubles et triples intégrales sont un peu compliquées à obtenir sans lui (en l'utilisant, on a les commandes `\iint` et `\iiint` qui donnent \iint et \iiint).

C'est donc une bonne idée de toujours charger **amsmath** lorsqu'on veut écrire des maths, ou mieux encore de toujours charger **mathtools** qui charge **amsmath** et l'améliore (nous chargerons donc **mathtools** à la place de **amsmath**).

9.2. Symboles

Cette partie du tutoriel va recenser quelques symboles parmi les plus utilisés. Il est possible (et même conseillé) de la survoler rapidement lors de la première lecture, juste pour voir quels symboles sont disponibles. En effet, il sera toujours possible de s'y référer (ou de se référer à toute autre ressource) pour trouver comment écrire un symbole lorsqu'on en a besoin. C'est en pratiquant que les commandes des symboles les plus utilisées seront retenues, il ne sert à rien de les apprendre.

9.2.1. Les lettres

9.2.1.1. Les lettres grecques

Commençons par voir comment écrire des lettres grecques. Elles sont obtenues avec des commandes très simples puisque le nom de la commande est le nom de la lettre voulue (`\gamma` donne γ et `\Gamma` donne Γ). Cependant, les lettres dont la graphie est identique à celles de l'alphabet latin n'ont pas de commande et sont obtenues avec la lettre latine (par exemple, la lettre *omicron* est obtenue avec `o`).

Résultat obtenu	Commandes
$\alpha, \beta, \gamma, \delta, \epsilon, \zeta$	<code>\$\alpha\$, \$\beta\$, \$\gamma\$, \$\delta\$, \$\epsilon\$, \$\zeta\$</code>
$\eta, \theta, \iota, \kappa, \lambda, \mu$	<code>\$\eta\$, \$\theta\$, \$\iota\$, \$\kappa\$, \$\lambda\$, \$\mu\$</code>
$\nu, \chi, \omicron, \pi, \rho, \sigma$	<code>\$\nu\$, \$\chi\$, \$o\$, \$\pi\$, \$\rho\$, \$\sigma\$</code>
$\tau, \upsilon, \phi, \chi, \psi, \omega$	<code>\$\tau\$, \$\upsilon\$, \$\phi\$, \$\chi\$, \$\psi\$, \$\omega\$</code>

Résultat obtenu	Commandes
$A, B, \Gamma, \Delta, E, Z$	<code>\$A\$, \$B\$, \$\Gamma\$, \$\Delta\$, \$E\$, \$Z\$</code>
$H, \Theta, I, K, \Lambda, M$	<code>\$H\$, \$\Theta\$, \$I\$, \$K\$, \$\Lambda\$, \$M\$</code>
N, X, O, Π, P, Σ	<code>\$N\$, \$X\$, \$O\$, \$\Pi\$, \$P\$, \$\Sigma\$</code>
$T, \Upsilon, \Phi, \chi, \Psi, \Omega$	<code>\$T\$, \$\Upsilon\$, \$\Phi\$, \$\chi\$, \$\Psi\$, \$\Omega\$</code>

De plus, certaines lettres minuscules ont une variante. En particulier, φ obtenue avec `\varphi` et ε obtenue avec `\varepsilon` sont souvent préférés à `\phi` et `\epsilon`. En fait, souvent, on redéfinit les commandes `\epsilon` et `\phi` pour les remplacer par `\varphi` et `\varepsilon`.

Résultat obtenu	Commandes
$\varepsilon, \vartheta, \varrho$	<code>\$\varepsilon\$, \$\vartheta\$, \$\varrho\$</code>
$\varphi, \varpi, \varsigma$	<code>\$\varphi\$, \$\varpi\$, \$\varsigma\$</code>



La commandes `\sum` ne doit pas être utilisée pour obtenir la lettre *sigma*. Elle ne doit être utilisée **que** comme symbole de sommation. Même chose pour la commande `\prod` et la lettre *pi*.

9.2.1.2. Autres lettres

Voyons d'autres commandes pour obtenir des symboles utiles, comme le « h » barrée utilisé pour représenter la constante de Planck réduite.

Symbole	Commande	Commentaire
\aleph	<code>\aleph</code>	Première lettre de l'alphabet hébreu
\Re et \Im	<code>\Re</code> et <code>\Im</code>	Partie réelle et imaginaire d'un nombre complexe
∇	<code>\nabla</code>	Nabla
∂	<code>\partial</code>	Symbole de la dérivée partielle
\hbar	<code>\hbar</code>	Constante de Planck réduite

9.2.2. Opérateurs binaires

Nous disposons également d'un grand nombre d'opérateurs. Pour commencer, voyons les opérateurs binaires, c'est-à-dire les symboles utilisés pour représenter une opération entre deux expression (le symbole `+` est par exemple l'opérateur binaire de l'addition et `-` est celui de la soustraction).

Symbole	Commande	Commentaire
\pm et \mp	<code>\pm</code> et <code>\mp</code>	Plus ou moins
\times	<code>\times</code>	Multiplication (ne pas utiliser <code>x</code> ou <code>*</code>)
\div	<code>\div</code>	Division
\circ	<code>\circ</code>	Composition de fonctions
\cdot	<code>\cdot</code>	Produit scalaire
\wedge	<code>\wedge</code>	Produit vectoriel
\cup	<code>\cup</code>	Union
\cap	<code>\cap</code>	Intersection
\oplus	<code>\oplus</code>	Somme directe
\setminus	<code>\setminus</code>	Différence de deux ensembles (ne pas confondre avec <code>\backslash</code>)

II. Compléter vos documents

\otimes	<code>\otimes</code>	Produit tensoriel
-----------	----------------------	-------------------

Il existe plusieurs autres opérateurs binaires, mais nous avons vu les plus utiles ici.

Notons que certains de ces opérateurs eux possèdent également une version à taille variable qui permet d'indicer (comme `\sum` ou `\prod`). Nous avons alors accès aux commandes suivantes.

Résultat obtenu	Commande
$\bigcap_{i=0}^n E_i$	<code>\bigcap_{i = 0}^n E_i</code>
$\bigcup_{i=0}^n E_i$	<code>\bigcup_{i = 0}^n E_i</code>
$\bigoplus_{i=0}^n E_i$	<code>\bigoplus_{i = 0}^n E_i</code>
$\bigwedge_{i=0}^n x_i$	<code>\bigwedge_{i = 0}^n x_i</code>
$\bigotimes_{i=0}^n x_i$	<code>\bigotimes_{i = 0}^n x_i</code>

9.2.3. Relations binaires

Nous avons ensuite des symboles pour représenter des relations binaires, c'est-à-dire les symboles pour indiquer une relation entre deux éléments (le symbole `=` indique par exemple une relation d'égalité, `<` la relation « inférieur strict à » et `>` la relation « supérieur strict à »).

Symbole	Commande	Commentaire
\leq	<code>\leq</code>	Inférieur ou égal à
\geq	<code>\geq</code>	Supérieur ou égal à
\ll	<code>\ll</code>	Très inférieur à
\gg	<code>\gg</code>	Très supérieur à
\prec	<code>\prec</code>	
\succ	<code>\succ</code>	
\preceq	<code>\preceq</code>	
\succeq	<code>\succeq</code>	
\approx et \simeq	<code>\approx</code> et <code>\simeq</code>	Environ égal à
\sim	<code>\sim</code>	Équivalent à (ou suit une loi en probabilités)
\equiv	<code>\equiv</code>	Congru à (ou équivalent à)

II. Compléter vos documents

	<code>\$\mid\$</code>	Divise (ne pas confondre avec $\$ \$$)
\in	<code>\$\in</code>	Appartient à
\ni	<code>\$\ni</code>	Est un élément de
\subset	<code>\$\subset\$</code>	Est inclus dans
\supset	<code>\$\supset\$</code>	Contient
\subseteq	<code>\$\subseteq\$</code>	Inclusion non stricte
\supseteq	<code>\$\supseteq\$</code>	Contient ou est égal
\parallel	<code>\$\parallel\$</code>	Est parallèle à
\perp	<code>\$\perp\$</code>	Est perpendiculaire à

Nous pouvons obtenir la version négative de ces symboles en utilisant la commande `\not` juste avant la commande pour le symbole. Ainsi, avec `\not\in`, nous obtenons \notin et avec `\not >` nous obtenons \nlessgtr .

Notons de plus que nous avons la commande `\neq` pour obtenir \neq (équivalente à `\not =` et la commande `\notin` pour obtenir \notin (équivalente à `\not \in`).

9.2.4. Logiques et ensembles

Il existe plusieurs commandes liées aux ensembles et à la logique.

Symbole	Commande	Commentaire
\emptyset	<code>\$\emptyset\$</code>	Ensemble vide
\exists	<code>\$\exists\$</code>	Quantificateur existentiel, il existe
\forall	<code>\$\forall\$</code>	Quantificateur universel, pour tout
\neg	<code>\$\neg\$</code>	Non logique
\wedge	<code>\$\wedge\$</code>	Et logique (<i>logical and</i>)
\vee	<code>\$\vee\$</code>	Ou logique (<i>logical or</i>)
\implies	<code>\$\implies\$</code>	Implication
\impliedby	<code>\$\impliedby\$</code>	Implication
\iff	<code>\$\iff\$</code>	Équivalence
\rightarrow	<code>\$\rightarrow\$</code>	Vers, dans

Notons que la commande `\land` est strictement équivalente à la commande `\wedge`. Cependant, elles n'ont pas la même sémantique (leur nom le montre bien) et `\land` doit être préféré pour la logique et `\wedge` pour les produits vectoriels.

9.2.5. Délimiteurs

Les délimiteurs sont les symboles comme les parenthèses qui servent à... Délimiter une expression en regroupant ses termes. En écrivant `$3 \times (x + 2)$`, j'utilise des délimiteurs. Voici la liste des délimiteurs utilisables.

Résultat obtenu	Commande
(x)	<code>\$ (x) \$</code>
$[x]$	<code>\$ [x] \$</code>
$\{x\}$	<code>\$ \{ x \} \$</code>
$\langle x \rangle$	<code>\$ \langle x \rangle \$</code>
$\lceil x \rceil$	<code>\$ \lceil x \rceil \$</code>
$\lfloor x \rfloor$	<code>\$ \lfloor x \rfloor \$</code>
$ $	<code>\$ \vert \$</code> ou <code>\$ \$</code>
\parallel	<code>\$ \Vert \$</code> ou <code>\$ \parallel \$</code>



Les commandes `\mid` et `\vert`, tout comme `\Vert` et `\parallel` sont différentes (`$a \mid b$` donne $a \mid b$ et `$a | b$` donne $a|b$). Cela est dû au fait qu'il s'agit de deux types d'objets différents. `\mid` représente une relation, et `|` représente un délimiteur. Il faut veiller à utiliser le bon objet au bon moment.

Ainsi, nous écrirons une norme en utilisant `\Vert` (et mieux, nous créerons une commande pour la norme pour avoir un code plus sémantique), mais nous utiliserons `\parallel` pour indiquer une relation entre deux expressions.

9.2.5.1. Taille des délimiteurs

Lorsque nous plaçons des délimiteurs autour d'un élément un peu grand (une fraction par exemple), le résultat n'est pas très joli :

$$\left(\frac{1}{2}\right).$$

Heureusement, il est possible d'obtenir des délimiteurs de taille adaptée à leur contenu grâce aux primitives `\left` et `\right`. Elles vont de pair et prennent en paramètre un délimiteur dont elles adaptent la taille à ce qui se trouve entre elles. On obtient alors

$$\left(\frac{1}{2}\right)$$

avec le code qui suit.

```
1 \[ \left( \frac{1}{2} \right) \].
```


II. Compléter vos documents

On peut associer des délimiteurs différents (accolades et parenthèses par exemple) et leur sens n'a pas d'importance. On peut alors obtenir

$$\left. \frac{1}{2} \right\}$$

en écrivant ce code.

```
1 \[ \left] \frac{1}{2} \right\} \]
```

Il existe également un délimiteur qui n'affiche rien, il s'agit de `.`. Son but est de permettre de n'avoir qu'un délimiteur affiché. En effet, puisqu'à chaque commande `\left` doit correspondre une commande `\right`, pour n'afficher qu'un seul délimiteur, il suffit d'utiliser `.`. On a alors

$$\frac{1}{2} \left| \right.$$

avec ce code.

```
1 \[ \left. \frac{1}{2} \right| \]
```

9.2.6. Flèches

Nous disposons de plusieurs commandes pour composer des flèches. Plutôt que de juste recenser ces différentes flèches, nous allons voir comment retenir les noms des commandes.

9.2.6.1. Flèches normales

La première chose à savoir pour les flèches normales est que tous les noms des commandes se terminent par `\arrow`.

Ensuite, en associant plusieurs mots, on décrit la flèche comme on le veut.

- Juste avant `arrow`, on donne la direction de la flèche qui est `up`, `down`, `left` et `right` (on peut mettre des flèches aux deux extrémités avec `updown` ou `leftright`).
- En mettant la première lettre de la commande en majuscule, on obtient une flèche double.

Symbole	Commande
\leftarrow et \Leftarrow	<code>\leftarrow</code> et <code>\Leftarrow</code>
\rightarrow et \Rightarrow	<code>\rightarrow</code> et <code>\Rightarrow</code>
\uparrow et \Uparrow	<code>\uparrow</code> et <code>\Uparrow</code>
\downarrow et \Downarrow	<code>\downarrow</code> et <code>\Downarrow</code>
\leftrightarrow et \Leftrightarrow	<code>\leftrightarrow</code> et <code>\Leftrightarrow</code>
\updownarrow et \Updownarrow	<code>\updownarrow</code> et <code>\Updownarrow</code>

De plus, en préfixant le nom des commandes de flèches horizontales par `long`, on obtient une flèche plus longue (dans le cas, où on veut une flèche double, c'est toujours la première lettre de

II. Compléter vos documents

la commande, le « l » ici, qui doit être mise en majuscule.

Symbole	Commande
\longleftarrow et \Leftarrow	<code>\backslashlongleftarrow</code> et <code>\backslashLongleftarrow</code>
\longrightarrow et \Rightarrow	<code>\backslashlongrightarrow</code> et <code>\backslashLongrightarrow</code>
\longleftrightarrow et \Leftrightarrow	<code>\backslashlonglefttrightarrow</code> et <code>\backslashLonglefttrightarrow</code>

9.2.6.2. Flèches diagonales

Nous disposons également de quelques flèches supplémentaires. Pour commencer les flèches en diagonale. Leur nom se retient également, il se base sur les points cardinaux en anglais (à savoir *north*, *east*, *west* et *south*). Le nom se compose simplement des initiales de la direction suivie de `arrow`.

Symbole	Commande
\nearrow , \searrow , \nwarrow et \swarrow	<code>\backslashnearrow</code> , <code>\backslashsearrow</code> , <code>\backslashnwarrow</code> et <code>\backslashswarrow</code>

9.2.6.3. Autres flèches

Et nous avons encore d'autres flèches. Notons notamment la flèche \mapsto obtenue avec `\backslash mapsto` (et sa version plus longue \longmapsto obtenue avec `\backslash longmapsto`) souvent utilisée lors de la définition de fonctions.

Au niveau des flèches horizontales, nous avons également les flèches recourbées dont le nom de la commande est composée de cette manière : `\backslash hook<direction>arrow`.

Symbole	Commande
\hookleftarrow et \hookrightarrow	<code>\backslashhookleftarrow</code> et <code>\backslashhookrightarrow</code>

Et enfin, toujours du côté des flèches horizontales, nous avons les flèches « harpons », dont le nom de la commande est composée de cette manière : `\backslash <direction_flèche>harpoon<direction_harpon>`.

Symbole	Commande
\leftharpoonup et \Leftharpoonup	<code>\backslashleftharpoonup</code> et <code>\backslashLeftharpoonup</code>
\leftharpoonright et \Leftharpoonright	<code>\backslashleftharpoondown</code> et <code>\backslashLeftharpoondown</code>
\rightharpoonright	<code>\backslashrightharpoonright</code>

L'une des choses les plus difficiles quand on commence à écrire des mathématiques avec LaTeX est d'utiliser la bonne commande. Par exemple, il ne faut pas confondre `|` et `\mid` . De plus,

II. Compléter vos documents

même si `\to` et `\rightarrow` donnent le même résultat, dans le cas de définition de fonctions ou de limites, il vaut mieux utiliser `\to` qui est alors plus sémantique.

Pour savoir quelle commande utiliser, on peut se demander quel type de symbole on utilise. Si on a besoin d'un délimiteur, on ne va pas utiliser la commande `\mid` qui permet d'obtenir un opérateur de relation.

Certains *packages* donnent accès à plusieurs autres symboles. Le *package* **amssymb** en particulier donne accès à une grande liste de symboles et est, de même que `amsmath`, quasiment incontournable lorsqu'on écrit des mathématiques. Il propose par exemple la commande `\complement` permet d'obtenir le symbole \complement pour noter le complémentaire d'une partie ou encore la commande `\varnothing` qui permet d'obtenir le symbole \emptyset pour l'ensemble vide alors qu'`\emptyset` donne \emptyset .

C'est donc une bonne idée d'également charger `amssymb` lorsque nous écrivons des maths. Nous chargerons donc `mathtools` et `amssymb` lorsque ce sera le cas.

9.3. Environnements mathématiques et autres commandes utiles

9.3.1. Environnements

Nous disposons de plusieurs environnements mathématiques pour par exemple numéroter les équations ou écrire des systèmes.

9.3.1.1. `equation`

L'environnement `equation` est le plus simple. Il agit comme `displaymath`, mais en plus numérote la formule (la numérotation est automatique). On a alors une équation numérotée avec le code qui suit.

```
1 \begin{equation}
2   f(x) = 3x + 2
3 \end{equation}
```



L'environnement `equation` s'utilise en dehors du mode mathématique. Dans le cas contraire, nous obtiendrions une erreur. Ce code est donc erroné.

```
1 \[
2 \begin{equation}
3   f(x) = 3x + 2
4 \end{equation}
5 \]
```

II. Compléter vos documents

9.3.1.2. split

L'environnement `split`, lui, nous permet d'écrire des équations sur plusieurs lignes. Pour indiquer que l'on veut passer à la ligne suivante, il nous suffit d'écrire `\\`. On obtient alors

$$f(x) = x^5 + 2x^4 + 3x^3 \\ + 4x^2 + 5x + 6$$

avec ce code.

```
1 \[
2 \begin{split}
3   f(x) = x^5 + 2x^4 + 3x^3 \\
4           + 4x^2 + 5x + 6
5 \end{split}
6 \]
```



Contrairement à `equation`, quand on utilise `split`, on doit être en mode mathématique. De plus, l'équation n'est pas numérotée.

L'environnement `split` permet aussi d'aligner les lignes comme on le veut en utilisant le symbole `&` (l'alignement se fera alors au niveau de ce symbole).

$$f(x) = x^5 + 2x^4 + 3x^3 \\ + 4x^2 + 5x + 6$$

est alors obtenu avec ce code.

```
1 \[
2 \begin{split}
3   f(x) = x^5 &+ 2x^4 + 3x^3 \\
4           &+ 4x^2 + 5x + 6
5 \end{split}
6 \]
```

Notons que l'aération et la présentation du code restent un choix ; nous ne sommes pas obligés d'aller à la ligne après `\\` et d'aligner effectivement les `&` dans le code. Le code suivant est strictement équivalent à celui que nous venons d'écrire (mais est moins lisible).

```
1 \[
2 \begin{split} f(x) = x ^ 5 & + 2 x^ 4 + 3x ^3 \\&+4x^2 +5x+6
3 \end{split}
4 \]
```

Nous ne verrons pas plus de choses sur les environnements ici, mais il nous faut savoir que plusieurs autres environnements existent, permettant par exemple d'écrire des matrices ou des tableaux.

Pour plus d'informations sur les environnements mathématiques, nous pouvons aller voir [ce](#)

[tutoriel](#) et plus particulièrement son [deuxième chapitre](#). Il présente notamment les environnements proposés par les *packages* `amsmath` et `mathtools` (oui, encore eux) et la gestion des références aux équations.

9.3.2. Théorèmes

Il n'est pas rare de vouloir écrire des théorèmes lorsqu'on écrit des documents scientifiques, et avec LaTeX, nous pouvons en écrire facilement. Voyons d'abord un exemple de son utilisation que nous allons ensuite commenter.

```
1 \documentclass[12pt, french]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
4 \usepackage{mathtools} % Charge amsmath.
5 \usepackage{amssymb}
6 \usepackage{babel}
7
8 \newtheorem{theorem}{Théorème}[section]
9 \newtheorem{proposition}{Proposition}[subsection]
10
11 \begin{document}
12   \section{Théorème}
13   Voici un premier théorème.
14   \begin{theorem}[Exemple de théorème]
15     Voici notre premier exemple de théorème. Quel résultat
16     obtiendra-t-on ?
17   \end{theorem}
18
19   Allons maintenant dans une sous-section.
20
21   \subsection{Théorème et proposition}
22   \begin{theorem}
23     Et on fait quand même un second exemple pour être sûr.
24     Après tout qui sait.
25   \end{theorem}
26
27   \begin{proposition}[Exemple de proposition]
28     Les propositions aussi c'est bien. Même si ça ne vaut
29     pas un bon théorème...
30   \end{proposition}
31 \end{document}
```

Pour écrire un théorème, il faut d'abord créer un type de théorème avec la commande `\newtheorem`. Elle s'emploie comme suit.

```
1 \newtheorem{<nom>}{<text>}[<compteur>]
```

Cette commande nous permet alors d'utiliser l'environnement `theorem` qui crée un théorème.

```
1 \begin{<nom>}[Exemple de théorème]
2   Voici un théorème
3 \end{<nom>}
```

Le théorème est alors affiché de la manière qui suit.

- Le deuxième argument de `\newtheorem` est imprimé.
- Un numéro est imprimé. En argument facultatif de `\newtheorem` nous donnons un nom associé à un compteur. Cela permet d'indiquer que l'on veut que la numérotation recommence à chaque fois que l'on crée une nouvelle section par exemple. De plus, le numéro du théorème sera alors précédé de celui de la section.
- L'argument optionnel de l'environnement est affiché (si on l'a passé) entre parenthèses. Généralement, il s'agit du nom du théorème ou de son auteur.
- Le contenu du théorème est affiché en italique.

Grâce à cela, nous pouvons créer autant d'environnements que nécessaires, pour les théorèmes, les définitions, les lemmes, etc. Ils auront tous leur propre compteur. Et on peut également y faire référence comme le montre le code qui suit.

```
1 \begin{proposition}[Exemple de proposition]\label{prop:exemple}
2   Un exemple de proposition qu'on utilise pour illustrer les
   références.
3 \end{proposition}
4 On fait référence à la proposition \ref{prop:exemple}.
```

i

Pour bénéficier de plus de fonctionnalités, nous pourrions nous renseigner sur les packages qui permettent de gérer les théorèmes. Le *package* `ntheorem` est par exemple présenté dans [ce tutoriel](#) .

9.3.3. Autres commandes utiles

Et il reste pleins d'autres commandes très utiles que nous n'avons pas vues. Bien sûr, nous ne pourrions pas toutes les voir, mais nous pouvons quand même présenter quelques fonctionnalités utiles.

9.3.3.1. Accolades horizontales

Nous obtenons des accolades horizontales qui s'étendent au dessus (respectivement en dessous) d'une formule, grâce à la commande `\overbrace` (respectivement `\underbrace`). Elles prennent en paramètre le texte qui doit être affublé de l'accolade.

$$\overbrace{(2+3)} \times \underbrace{(1+2)} = 2 + 4 + 3 + 6 = 15$$

s'obtient avec ce code.

II. Compléter vos documents

```
1 \[ \overbrace{(2 + 3)} \times \underbrace{(1 + 2)} = 2 + 4 + 3 + 6
   = 15 \]
```

De plus, en utilisant les symboles $\overset{\wedge}$, et $\underset{_}$, on peut écrire au dessus et en dessous des accolades (la police est alors plus petite).

$$\overset{=5}{(2 + 3)} \times \underset{=3}{(1 + 2)} = 2 + 4 + 3 + 6 = 15$$

s'obtient avec ce code.

```
1 \[
2   \overbrace{(2 + 3)}^{\{= 5\}} \times \underbrace{(1 + 2)}_{\{= 3\}}
3   = 2 + 4 + 3 + 6 = 15
4 \]
```



$\overset{\wedge}$ peut également s'utiliser avec `\underbrace`, de même que $\underset{_}$ peut s'utiliser avec `\overbrace`.

9.3.3.2. Superposition de symboles

Il peut être très utile de mettre un symbole au dessus ou en dessous d'un autre. Le mieux pour cela est d'utiliser les commandes `\overset` et `\underset` du *package* `amsmath`. Elles permettent d'obtenir

$$x \overset{f}{\mapsto} f(x), y \underset{f}{=} x$$

en écrivant ce code.

```
1 \[ x \overset{f}{\longmapsto} f(x), y \underset{f}{=} x \]
```

Mais avant de chercher à utiliser ces commandes, il vaut mieux chercher si le symbole que l'on veut obtenir n'existe pas. Par exemple, il est inutile d'utiliser `\overset{.}{=}` pour obtenir \doteq puisque la commande `\doteq` permet déjà d'obtenir ce symbole.

Ce chapitre très long nous présente les bases de l'écriture mathématique. Pour le compléter, il est conseillé d'aller consulter d'autres ressources, notamment [celle-ci](#) qui présente les outils mathématiques plus profondément (création de commandes personnalisées, présentation des environnements mathématiques les plus utiles, etc.).

Tout comme Rome ne s'est pas faite en un jour, nous n'apprendrons pas à bien écrire des mathématiques en un jour, mais petit à petit, nous nous améliorerons et écrirons facilement et efficacement.

10. De nouvelles commandes

Dans ce chapitre, nous allons apprendre à créer nos propres commandes, puis nous verrons où trouver des classes et des *packages* déjà faits et comment apprendre à les utiliser. Avec nos propres commandes, nous pourrions avoir du code plus facile à écrire et surtout plus lisible.

Par exemple, si nous faisons plusieurs fois des citations, il pourrait être judicieux d'avoir une commande de citation qui prend en paramètre la citation et son auteur et la met en forme. Ainsi, notre code sera plus lisible grâce à l'utilisation de cette commande, et en plus nous sommes assurés d'avoir un style de citation uniforme dans tout le document.

10.1. Créer ses commandes

La première étape est de voir comment définir de nouvelles commandes et de nouveaux environnements.

10.1.1. Créer une commande

Pour définir une nouvelle commande, nous utilisons la commande `\newcommand`.

```
1 \newcommand{\nom_commande}[nb_arguments]{code_de_la_commande}
```

On lui donne le nom de la commande à définir, son nombre d'arguments (0 par défaut) et le code de la commande.

Pour illustrer son fonctionnement, nous allons prendre l'exemple d'une commande de citation, qui prend en paramètre le nom de l'auteur et la citation, et la met en forme entre guillemets.

Voici comment nous pourrions définir cette commande.

```
1 \newcommand{\citer}[2]{\og #2 \fg (\bsc{#1})}
```

Ici, nous définissons une commande `\citer`, qui prend 2 arguments. Dans le code de la commande, nous faisons référence au n^e argument en écrivant `#n`. Ainsi, cette commande écrit son deuxième argument entre guillemets, suivi de son premier argument entre parenthèses.

i

Généralement, les définitions de commandes sont placées dans le préambule du document.

Essayons notre commande.

```
1 \documentclass[12pt,french]{article}
2 \usepackage[utf8]{inputenc}
3 \usepackage[T1]{fontenc}
```


II. Compléter vos documents

```
4 \usepackage{babel}
5
6 \newcommand{\citer}[2]{\log #2 \fg{ } (#1)}
7
8 \begin{document}
9   Pouvons-nous dire qu'\citer{Machiavel}{une guerre est juste
10     quand elle nécessaire} ?
11 \end{document}
```



Nous ne pouvons pas définir une commande qui existe déjà, sous peine d'obtenir l'erreur « command <\nom_commande> already defined ».

10.1.1.1. Commandes courtes

Notre commande n'est pas très adaptée pour les citations sur plusieurs paragraphes. Nous ne voudrions pas l'utiliser de cette manière par exemple.

```
1 \citer{auteur}{Une citation longue. En fait
2 sur plusieurs paragraphes. Avec des listes.
3 \begin{itemize}
4   \item Un item.
5   \item Un deuxième.
6   \item Un dernier pour la route.
7 \end{itemize}
8
9 Un autre paragraphe de la citation.
10 }
```

Pourtant ce code est légal et ne donnera pas d'erreurs. Pour régler ce problème, il nous suffit de définir une commande courte. Et pour cela il n'y a presque rien à changer : il faut juste utiliser la version étoilée de `\newcommand`.

```
1 \newcommand*\citer[2]{\log #2 \fg{ } (#1)}
2
3 \begin{document}
4   \citer{auteur}{Une citation longue. En fait
5   sur plusieurs paragraphes. Avec des listes.
6   \begin{itemize}
7     \item Un item.
8     \item Un deuxième.
9     \item Un dernier pour la route.
10  \end{itemize}
11
12   Un autre paragraphe de la citation.
13 }
14 \end{document}
```

II. Compléter vos documents

Cette fois, nous obtenons une erreur : « paragraph ended before \citer was complete ».

10.1.2. Créer un environnement

Pour créer un environnement, nous pouvons tout simplement créer une commande `\nom` et une commande `\endnom` (rappelons-nous [ceci](#) [↗](#)). Cependant, il existe également une commande `\newenvironment` qui s'utilise de cette manière.

```
1 \newenvironment{nom_environnement}[nb_arguments]{code_de_début}{code_de_fin}
```

En gardant l'exemple de la citation, créons un environnement de citation qui imprime « auteur a dit : » suivi de la citation dans un environnement `quotation`.

```
1 \newenvironment{citerlong}[1]{\bsc{#1} a dit :  
  \begin{quotation}}{\end{quotation}}
```

On peut alors écrire ce code.

```
1 \documentclass[12pt,french]{article}  
2 \usepackage[utf8]{inputenc}  
3 \usepackage[T1]{fontenc}  
4 \usepackage{babel}  
5  
6 \newenvironment{citerlong}[1]{\bsc{#1} a dit :  
  \begin{quotation}}{\end{quotation}}  
7  
8 \begin{document}  
9   \begin{citerlong}{Tartempion}  
10    Voici une citation longue.  
11  
12    Tellement longue qu'elle est sur plusieurs paragraphes.  
13  
14    Espérons que l'environnement saura distinguer les différents  
    paragraphes.  
15   \end{citerlong}  
16 \end{document}
```

Et on a bien le résultat attendu.

i

De même que pour les commandes, les définitions d'environnements sont généralement placées dans le préambule, et essayer de définir un environnement qui existe déjà donnera une erreur.

10.1.3. Redéfinition de commande et d'environnements

S'il n'est pas possible de définir une commande ou un environnement qui existe déjà, on peut cependant les redéfinir grâce aux commandes `\renewcommand` et `\renewenvironment`. Elles

II. Compléter vos documents

prennent les mêmes arguments que `\newcommand` et `\renewenvironment`.

Par exemple, si un jour nous voulons que la commande `\bsc` mettent les noms qu'on lui passe en argument en emphase plutôt qu'en petites capitales, nous la redéfinirons de cette manière.

```
1 \renewcommand{\bsc}[1]{\emph{#1}}
```

La redéfinition de commandes est très utile pour garder un texte sémantique. Comme nous l'avons déjà dit, si un jour nous voulons les mots importants en gras, nous redéfinirons la commande `\emph` plutôt que d'utiliser une commande de mise en gras pour chaque mot important. Cette solution, en plus d'être plus sémantique, est plus simple à mettre en place (si un jour nous en avons marre du gras, et voulons tout mettre en rouge, il nous suffira de changer la redéfinition).



La commande `renewcommand` a aussi une version étoilée qui agit de la même manière que la version étoilée de `\newcommand`. Ici, nous avons redéfini `\bsc` avec la version non étoilée de `\renewcommand`, donc elle est théoriquement utilisable avec plusieurs paragraphes (en pratique, elle ne l'est pas puisqu'elle utilise la commande `\emph` qui elle est courte).

10.1.3.1. Redéfinition récursive

Il arrive que l'on veuille redéfinir une méthode, mais en l'utilisant pour sa redéfinition. Un code de ce type en gros.

```
1 \renewcommand*{\emph}[1]{(\emph{#1} en emphase)}
2
3 \begin{document}
4   \emph{important}
5 \end{document}
```

Si nous essayons ceci, nous aurons l'erreur `TeX capacity exceeded`. En effet, ici, en appelant la commande `\emph`, on voit qu'elle utilise la commande `\emph`, donc on l'appelle, mais elle s'utilise donc on l'appelle... Et ce jusqu'à l'erreur.

Pour régler ce problème, il nous faut garder en mémoire l'ancienne commande `\emph` dans une autre commande, et appeler cette commande que nous venons de définir plutôt que l'ancienne.

```
1 \newcommand*{\oldemph}[1]{\emph{#1}}
2 \renewcommand*{\emph}[1]{(\oldemph{#1} en emphase)}
3
4 \begin{document}
5   \emph{important}
6 \end{document}
```



Mais ça ne marche pas ! L'erreur est toujours présente !

Le problème, c'est qu'en définissant `\oldemph`, on est en train de dire (en gros) « quand tu croises `odlemph`, tu devras appeler `\emph` », et donc voici ce qui arrive quand on appelle `\emph` :

II. Compléter vos documents

elle utilise `\oldemph`, donc on appelle `\oldemph`, mais celui-ci demande d'appeler `\emph`, et on retombe dans ce cercle vicieux. Voici un exemple illustrant ce phénomène.

```
1 \newcommand*\exemple}[1]{\emph{#1}}
2 \renewcommand*\emph}[1]{\bsc{#1}}
3
4 \begin{document}
5   \exemple{important} % Ici, important sera écrit en utilisant la
6   commande \bsc.
7 \end{document}
```

Pour régler ce problème, nous allons utiliser la primitive Tex `\let`. Avec `\let\oldemph\emph`, on indique à Tex que `\oldemph` doit prendre la valeur qu'a `\emph` au moment où est écrite la ligne.

i

En fait, ce comportement est à opposer à celui d'une autre primitive Tex, la primitive `\def`, puisque qu'avec `\def\oldemph\emph`, on indique à TeX qu'il faudra que `\oldemph` prenne la valeur de `\emph`, mais à la ligne d'utilisation (et donc si depuis la commande `\emph` a été modifiée, cela sera pris en compte).

Tout ceci nous permet de finalement faire un exemple correct.

```
1 \let\oldemph\emph
2 \renewcommand*\emph}[1]{(\oldemph{#1} en emphase)}
3
4 \begin{document}
5   \emph{important}
6 \end{document}
```

Ici, nous voyons comment créer une commande avec un argument optionnel.

👁 Contenu masqué n°7

10.2. Le CTAN, découverte de classes et de packages

La création de commandes c'est bien. Pouvoir profiter de commandes déjà écrites par d'autres personnes, c'est mieux. Ces commandes sont regroupées dans des *packages* et certaines personnes ont même créé des classes alternatives (par exemple pour écrire des thèses, des lettres, des journaux, etc.). Il est donc intéressant d'avoir ces *packages* (et ils sont vraiment très nombreux) à notre disposition.

Le **CTAN** [se décrit](https://www.ctan.org/) lui-même comme *the central place for all kinds of material around TeX* c'est-à-dire le site où l'on retrouve toutes sortes de choses autour de TeX (et aussi de LaTeX). On peut y trouver une bonne quantité de *packages*.

La page d'accueil du **CTAN** peut sembler austère à première vue. Si nous cherchons par exemple un *package* pour écrire du pseudo-code, nous pouvons naviguer à travers la liste des sujets (« Browse » → « Topics »). Là, nous choisissons la lettre « P », puis la section pseudo-code et

II. Compléter vos documents

nous avons une liste de *packages* permettant d'écrire du pseudo-code.



Mais comment savoir quel *package* prendre ? Et comment savoir dans quel *topic* trouver ce qui nous intéresse.

Les noms des sujets sont généralement assez parlant, et ce même pour quelqu'un qui n'est pas habitué à l'anglais. Nous trouvons par exemple les sujets `cv`, `lettre` ou encore `colour`.

Pour choisir un *package* parmi tous ceux qui font ce que l'on désire, il n'y a pas de méthode miracle ; le mieux reste de regarder les avis (voir de les demander). Néanmoins, il faudra parfois tester avant de trouver le *package* que l'on veut.

Bien sûr, la fonction de recherche du site sert également. Si par exemple, nous cherchons des *packages* permettant de formater des liens, nous pouvons `rechercher url` [↗](#). La page de résultat nous liste des *packages* ayant un lien avec notre recherche et nous pouvons alors nous renseigner à propos de ces *packages* et trouver lequel nous convient.



La recherche peut être personnalisée de plusieurs manières. La colonne « *Tips for the search* » nous aide à construire des recherches plus précises.

Regardons l'un des résultats de notre recherche, `le package url` [↗](#). Sur la page du *package* nous avons une petite description de ce qu'il fait, quelques informations (licence, sources, version, etc.), un lien pour le télécharger, des suggestions de *packages* et même une note. Néanmoins, l'information qui va le plus nous intéresser est la documentation quand elle est présente.

Certaines classes et certains *packages* ne sont pas présents sur le **CTAN**, mais sur d'autres sites (site de l'auteur, Github, etc.).

10.3. Utiliser les documentations

Quasiment toutes les classes et tous les *packages* que nous utilisons sont accompagnés d'une documentation. Celle-ci nous liste les choses à connaître sur le *package* (nom et arguments des différentes commandes, problèmes que l'on peut rencontrer, compatibilité ou incompatibilité avec d'autres *packages*, etc.). La documentation permet alors de savoir tout ça et donc de prendre en main un nouveau *package*.

Les documentations sont généralement en anglais, mais c'est de l'anglais technique assez simple à comprendre.



Mais comment trouver une documentation ?

La documentation d'un *package* est généralement disponible sur le **CTAN** (si ce *package* est disponible sur le **CTAN**) ou encore sur le site de l'auteur. En gros, la documentation sera trouvable sur la page Web du projet.

Néanmoins, le plus simple pour avoir accès à une documentation reste la commande `texdoc`. Ainsi, en tapant `texdoc babel` dans un terminal, le PDF de la documentation de `babel` est censé s'ouvrir. Ceci dépend cependant des choix faits à l'installation de la distribution LaTeX (on choisit d'installer ou non les documentations sur notre ordinateur).



Puisque ces documentations sont stockées sur notre ordinateur, elles ne sont pas forcément à jour (tout comme les *packages* que nous avons sur notre ordinateur).

Comme nous pouvons le voir en ouvrant la documentation de *babel*, elle est plutôt grosse (aux alentours de 150 pages). Pas d'inquiétudes ! C'est normal, *babel* est un gros *package* avec beaucoup d'options. La plupart des documentations font beaucoup moins de pages, et en fait, les documentations se lisent assez facilement une fois qu'on est habitué (oui, même celle de *babel*) car elles sont très bien documentées.

En fait, la plupart des documentations suivent un schéma somme toute assez classique. Tout d'abord, on retrouve une petite introduction qui indique l'utilité du *package*, ce qu'il apporte, ses avantages et désavantages (parfois par rapport aux autres *packages* existants), etc. Puis viennent généralement une présentation générale du *package*. Et finalement, les différentes commandes qu'il offre sont présentées de même que les options éventuelles du *package*, et bien sûr, on peut souvent trouver quelques exemples d'utilisation.

Il n'est pas rare de trouver également un récapitulatif des problèmes que l'on peut rencontrer avec le *package* (en particulier quand il peut être en conflit avec d'autres *packages* connus), et des améliorations qui sont à venir.

Bien sûr, en lisant des documentations, on s'habitue à en lire et à trouver les informations qui nous intéressent de manière efficace.

La documentation du *package* `url` pris en exemple plus haut est assez courte et peut être un bon premier pas pour apprendre à lire une documentation.

Ce petit chapitre nous aura introduit à la création de commandes et d'environnements et surtout nous aura présenté le très grand monde des *packages* du CTAN. Voici quelques *packages* souvent utilisés dans leurs domaines respectifs.

- **csquotes** qui permet de faire des citations (il propose par exemple une commande `\enquote` qui place juste des guillemets autour de son argument).
- **hyperref** pour créer des hyperliens et gérer les méta-données des PDFs créés.
- **biblatex** pour la gestion des bibliographies (avec `biber`).
- **tikz** pour dessiner et créer des graphiques et **pgfplots** qui utilise `tikz` et offre des outils de plus haut niveau.

Tout comme l'on devient forgeron en forgeant, c'est en utilisant LaTeX qu'on découvre les *packages* qui nous sont utiles et que l'on se forge petit à petit sa propre boîte à outils, que ce soit en termes de *packages* ou de commandes personnelles.

Contenu masqué

Contenu masqué n°7

Pour créer une commande avec un argument optionnel, nous allons utiliser le second argument optionnel de `\newcommand`. Nous mettons en premier argument optionnel le nombre total d'arguments de la commande que l'on veut créer, et la présence d'un second argument optionnel indique à LaTeX qu'il y a un argument optionnel. Cet argument optionnel sera alors l'argument `#1`.

II. Compléter vos documents

```
1 \newcommand\test[1][]{Argument optionnel (#1).}
```

Avec le code `\test \test[A]`, nous obtenons alors « Argument optionnel : (). Argument optionnel : (A) ». Notons que nous pouvons donner une valeur par défaut à l'argument optionnel. Il nous suffit pour cela de la donner dans le second argument optionnel de `\newcommand`. Ainsi, on écrit ce code.

```
1 \newcommand\test[1][X]{Argument optionnel (#1).}
```

Cette fois, le code `\test \test[A]` nous donne « Argument optionnel : (X). Argument optionnel : (A) ».



Puisque `#1` fait référence à l'argument optionnel, les autres arguments, ceux qui sont obligatoires commencent bien sûr à partir de `#2`.

Réécrivons `\test` avec deux arguments obligatoires qu'elle affiche (en plus de l'argument optionnel).

```
1 \newcommand\test[3][X]{Arguments : optionnel (#1), obligatoire 1  
  (#2), obligatoire 2 (#3).}
```

Avec `\newcommand`, nous ne pouvons avoir qu'un seul argument optionnel. Quelques astuces nous permettent d'outrepasser cette limite, mais nous n'allons pas les voir ici. Lorsque nous voudrons faire cela, et plus généralement lorsque nous voudrons créer de manière plus précise des commandes, nous nous tournerons vers des *packages* comme **xparse** qui nous facilitent la tâche.

[Retourner au texte.](#)

11. Mise en forme

Dans ce chapitre, nous allons voir différentes commandes de mises en forme. Nous allons notamment voir comment gérer la taille de notre texte et son alignement.

11.1. Le texte

Pour commencer, voyons comment changer l'apparence de notre texte.

11.1.1. Changement de taille

Commençons par voir comment changer la taille du texte. Nous disposons de plusieurs commandes à bascule pour changer la taille de notre texte.

Commande	Effet
<code>\tiny</code>	Minuscule
<code>\scriptsize</code>	Très petit
<code>\footnotesize</code>	Assez petit
<code>\small</code>	Petit
<code>\normalsize</code>	Normale (choisie dans le <code>\documentclass</code>)
<code>\large</code>	Grand
<code>\Large</code>	Assez Grand
<code>\LARGE</code>	Très grand
<code>\huge</code>	Énorme
<code>\Huge</code>	Géant

TABLE 11.2. – Les commandes de changement de taille.

Ainsi, on peut changer de taille d'un paragraphe à un autre de cette manière.

1	Ce texte est en taille normale.
2	
3	<code>\begin{Large}</code>
4	Dans ce paragraphe, nous écrivons dans une taille un peu plus grande.
5	<code>\end{Large}</code>
6	


```

7 On est {\tiny vraiment minuscule et là on} {\Large grandit d'un
   coup avant de}
8 revenir à notre taille normale.

```

Les tailles obtenues avec ces commandes sont définies par rapport à la taille choisie dans le préambule de notre document, `\normalsize` correspondant à cette taille (c'est donc la taille par défaut). Cela signifie que la taille obtenue avec `\Large` lorsque nous choisissons une taille de `11pt` pour notre document est plus petite que celle que nous aurions obtenu si nous avions choisi une taille de `12pt`. Ces tailles sont donc **relatives à la taille par défaut**.



Il nous faut réfléchir à notre taille par défaut **avant** de chercher à utiliser ces commandes. Par exemple, plutôt que d'utiliser `\small` sur tout un document avec une taille par défaut de `12pt`, nous ferions mieux de choisir une taille par défaut de `11pt` ou `10pt`.

11.1.2. Changement de fontes

Il est également possible de changer de fontes. Nous pouvons changer quatre paramètres.

- La famille (*family* en anglais) pour commencer correspond à la forme globale. Nous disposons par défaut de trois familles :
 - la famille **romane** est la famille par défaut ;
 - la famille à **écartement fixe** ou à **chasse fixe** correspond à celle où l'espacement entre les lettres est constant (on parle aussi de *typewriter* car l'espacement sur les machines à écrire est fixe) ;
 - la famille **sans empattements** (on parle aussi de famille **sans serif**).
- Le style (*shape* en anglais). Par défaut, il y en a trois :
 - le style italique ;
 - le style penché (à ne pas confondre avec le style italique) ;
 - le style en petites capitales.
- La graisse (*series* en anglais) correspond à l'épaisseur des traits. Par défaut, il y en a deux :
 - le gras ;
 - le texte normal.

Nous disposons de deux commandes pour chaque changement, une normale et une à bascule (de la même manière que nous disposons de `\em` et de `\emph`).

Commande	Commande à bascule	Fonte
<code>\textrm</code>	<code>\rmfamily</code>	roman
<code>\textsf</code>	<code>\sffamily</code>	sans serif
<code>\texttt</code>	<code>\ttfamily</code>	machine à écrire
<code>\textup</code>	<code>\upshape</code>	droit
<code>\textit</code>	<code>\itshape</code>	italique
<code>\textsl</code>	<code>\slshape</code>	penché

<code>\textsc</code>	<code>\scshape</code>	petites capitales
<code>\textmd</code>	<code>\mdseries</code>	graisse normale
<code>\textbf</code>	<code>\bfseries</code>	gras

TABLE 11.4. – Les commandes de changement de fonte.

Nous pouvons alors essayer ce code.

```

1  Voici du \textbf{gras} puis de l'\textit{italique}. Nous pouvons
   même avoir du
2  \textbf{\textit{gras en italique}}.
3
4  Voici du {\bfseries gras} puis de l'{\itshape italique}. Nous
   pouvons même avoir du
5  {\bfseries \itshape gras en italique}}.
6
7  Mais nous ne pouvons pas avoir des \textsc{\textit{petites
   capitales en italique}}.
8
9  Mais nous ne pouvons pas avoir des {\scshape \itshape petites
   capitales en italique}}.
10
11 \begin{scshape}
12   Voici un groupe de mots en petite capitale.
13
14   Un bon groupe de mots...
15 \end{scshape}

```

Nous ne pouvons pas utiliser deux familles, deux graisses ou deux styles en même temps (ce qui paraît assez normal), mais nous pouvons choisir simultanément une famille, une graisse et un style.



Dans certaines sources, nous pouvons voir l'utilisation des commandes `\bf`, `\it`, `\tt`. Ces commandes à bascules sont dépréciées, elles ne doivent pas être utilisées. Nous leur préféreront les commandes à bascules `\bfseries`, `\itshape`, ou encore `\ttfamily` qui sont meilleures sur plusieurs points (par exemple, `{\bf \it texte}`, contrairement `{\bfseries \itshape texte}` ne donne pas un texte gras en italique, mais un texte en italique).

11.1.2.1. Commandes à bascules ou pas ?

Néanmoins, il reste une question, faut-il préférer les commandes à bascules ou pas ?

Il est conseillé de ne pas utiliser les commandes à bascule à moins de ne vouloir changer la fonte sur plus d'un paragraphe (les commandes comme `\textit` sont des commandes courtes de sorte que leur argument ne peut pas contenir la commande `\par`). Nous allons donc privilégier les commandes normales, même si dans certains cas, utiliser les commandes à bascules sera plus

II. Compléter vos documents

judicieux. Notons que les commandes à bascules sont également utilisées fréquemment dans les déclarations de commandes.

Une des raisons pour laquelle il est conseillé de ne pas utiliser les commandes à bascules est qu'elles n'effectuent pas la **correction italique**. La correction italique consiste à rajouter un espace à la fin d'un groupe de mots en italique pour qu'il ne touche pas le mot suivant. Regardons la différence ici.

```
1 \textit{texte}texte, \textit{texte} texte.
2
3 {\itshape texte}texte, {\itshape texte} texte.
```

Les espaces après avoir utilisé `\itshape` sont plus petits que ceux après avoir utilisé `\textit`.

11.1.3. Choix de police

En plus de pouvoir choisir une taille et une fonte, nous pouvons bien sûr choisir une police. Cela passe généralement par l'utilisation d'un *package*. Par exemple, pour utiliser la police *Deja Vu Serif*, nous pouvons utiliser le *package* **DejaVuSansSerif**.

```
1 \usepackage[utf8]{inputenc}
2 \usepackage{DejaVuSerif}
3 \usepackage[T1]{fontenc}
4
5 \begin{document}
6   On a changé de police pour ce document.
7 \end{document}
```

Plusieurs polices pour LaTeX peuvent être trouvées sur [ce site](#) .

Ici, avec le *package* *DejaVuSerif*, on a changé la police du document en entier, et la plupart des *packages* agissent de la sorte. Néanmoins, certains d'entre eux offrent la possibilité de ne mettre qu'une partie du texte dans la police choisie. C'est par exemple le cas de la police *French Cursive* . Elle nous donne par exemple la commande `\textcursive` qu'on peut utiliser dans notre document.

```
1 \usepackage{frcursive}
2 \usepackage[T1]{fontenc}
3
4 \begin{document}
5   Une phrase. \textcursive{Une phrase avec French Cursive}.
6 \end{document}
```

Il est également possible de choisir la police *French Cursive* en police par défaut en chargeant le *package* avec l'option `default`.

i

Chaque *package* peut se comporter de manière différente et a différentes options. Pour savoir comment en utiliser un, il nous faudra alors consulter des exemples et la documentation.



Dans le premier chapitre [☞](#) nous avons parlé des moteurs XeTeX et LuaTeX qui permettent d'utiliser des polices différentes plus facilement.

11.2. Alignement du texte

Pour aligner du texte, nous disposons de trois environnements, `center`, `flushleft` et `flushright` qui nous permettent respectivement de centrer, d'aligner à gauche, et d'aligner à droite. Il n'y a pas grand chose à dire à propos de `center` et de `flushright`.

```
1 \begin{center}
2 Une phrase centrée.
3 \end{center}
4
5 \begin{flushright}
6 Une phrase alignée à droite.
7 \end{flushright}
```

Mais `flushleft` peut laisser un peu perplexe.



Pourquoi `flushleft` existe, il suffit de n'utiliser ni `center`, ni `flushright` et c'est pareil, non ?

Une première raison à cela est que l'on pourrait vouloir utiliser `flushleft` à l'intérieur d'un environnement comme `center`.

```
1 \begin{center}
2 Une phrase centrée.
3 \begin{flushleft}
4 À gauche.
5 \end{flushleft}
6 Centré.
7 \end{center}
```

Bien sûr, ici on pourrait tout à fait juste fermer l'environnement `center`, écrire « à gauche », et ouvrir un nouvel environnement `center`.

Mais en fait, l'alignement à gauche n'est pas exactement la même chose que le mode normal. En effet, lorsque nous alignons à gauche, les alinéas sont supprimés et le bloc de texte est vraiment bien aligné à gauche. Normalement, LaTeX essaie d'espacer les mots et de les couper pour que chaque ligne se termine à peu près à la même hauteur. Avec l'utilisation de ces environnements, il ne le fait plus. De plus, les mots ne sont pas coupés contrairement au texte normal.

```
1 \begin{flushleft}
2 Pour bien voir les effets de l'alignement à gauche, on prend une
   phrase
```

```
3 suffisamment longue, comme celle qu'on est en train d'écrire par
  exemple
4 où nous ne manquons pas de remarquer à quel point l'auteur a traîné
  à
5 longueur en écrivant des choses inutiles dans le seul but de faire
  cette
6 phrase prendre plusieurs lignes.
7 \end{flushleft}
8
9 Pour bien voir les effets de l'alignement à gauche, on prend une
  phrase
10 suffisamment longue, comme celle qu'on est en train d'écrire par
   exemple
11 où nous ne manquons pas de remarquer à quel point l'auteur a traîné
   à
12 longueur en écrivant des choses inutiles dans le seul but de faire
   cette
13 phrase prendre plusieurs lignes.
```

La différence est clairement visible.

11.3. Couleur

11.3.1. Du texte en couleur

Les couleurs ne sont pas gérées nativement par LaTeX, et pour en utiliser nous devons charger d'autres *packages*. Ici, nous allons voir l'utilisation du package **color**.

i

Pour plus de fonctionnalités liées aux couleurs, nous pourrions nous renseigner sur le package **xcolor** qui améliore le package **color** et est donc une alternative très avantageuse. Nous chargerons donc plutôt **xcolor**.

Avec ce *package*, nous pouvons utiliser la commande `\textcolor{<couleur>}{<texte>}` qui écrire `<texte>` dans la couleur `<couleur>`. Comme l'indique la documentation, le *package* prédéfinit les 19 couleurs suivantes.

```
1 red green blue cyan magenta yellow black
2 gray white darkgray lightgray brown lime
3 olive orange pink purple teal violet
```

Ainsi, avec le code suivant, nous redéfinissons `\emph` pour écrire les textes importants en rouge.

```
1 \usepackage{xcolor}
2 \renewcommand*{\emph}[1]{\textcolor{red}{#1}}
3
4 \begin{document}
```

II. Compléter vos documents

```
5 Un document avec un \emph{mot important}.
6 \end{document}
```

Là encore, nous disposons d'une commande à bascule, la commande `\color` qui prend en argument une couleur. Tout comme les commandes pour les changements de taille, nous veillerons à utiliser la commande appropriée en fonction de l'utilisation.

```
1 \begin{quote} \color{green}
2   Voici une citation en vert.
3 \end{quote}
4
5 \begin{quote}
6   Une autre citation... Mais cette fois en noir.
7 \end{quote}
8 \end{document}
```

11.3.2. Définir ses couleurs

Pour le moment, on ne dispose que de huit couleurs ce qui est assez peu (en fait, c'est faux, le *package* `xcolor` en définit plusieurs autres). Heureusement, nous avons la possibilité de définir nos propres couleurs grâce à la commande `\definecolor`. Elle prend en paramètre le nom que l'on veut donner à la couleur, le modèle que l'on veut utiliser pour la définir (par exemple le modèle RGB avec `rgb` ou un niveau de gris avec `gray`) et les spécifications de la couleur avec ce modèle.

i

Les spécifications de la couleur sont entre 0 et 1 (et pas entre 0 et 255). Ainsi, on pourrait définir un violet de cette manière.

```
1 \definecolor{violet}{rgb}{0.54, 0.17, 0.89}
```

Les trois composantes de la couleur sont séparées par des virgules, avec le modèle `gray`, il n'y a bien sûr qu'un seul nombre entre 0 et 1.

Notons que les commandes `\color` et `\textcolor` peuvent aussi prendre en paramètre la spécification d'une couleur à la place d'un nom de couleur si on donne le modèle en paramètre optionnel. Nous pouvons alors écrire ce code.

```
1 \begin{quote} \color[rgb]{0.2, 0.6, 0.2}
2   Voici une citation en vert.
3 \end{quote}
```

Cette méthode peut s'avérer utile dans le cas où on utilise une couleur une seule fois (et qu'on ne veut donc pas la définir pour une seule utilisation).

Nous pourrions regarder la documentation de `xcolor` pour obtenir plus d'informations sur les différents modèles de couleurs disponibles. Le modèle `named` en particulier peut être très intéressant. Dans cette documentation, nous verrons également les autres couleurs proposées.

Avec ce chapitre nous touchons (enfin) à la forme de nos documents. D'autres choses sont également possibles (nous pouvons par exemple encadrer du texte). Nous verrons cela dans de prochains chapitres.

i

Comment utiliser ces commandes

Le mieux est de ne pas les utiliser dans le corps du document, mais pour faire nos commandes et nos environnements. Si nous voulons les mots importants en gras, il vaudra mieux redéfinir `\emph` plutôt qu'utiliser `\textbf`. De même, si nous voulons centrer certains paragraphes pour une même raison X, il vaudra mieux définir un nouvel environnement.

Ainsi, le corps de nos document garde de la sémantique et il est plus simple de changer l'apparence (si nous voulons maintenant aligner ces paragraphes à droite, il y a juste à modifier l'environnement correspondant).

12. Images, tableaux et texte préformaté

Dans ce chapitre, nous allons apprendre à compléter nos documents en y rajoutant des images, des tableaux, ou encore des blocs de code.

12.1. Images

12.1.1. Le package `graphicx`

Pour insérer une image dans un document, nous allons utiliser le *package* **graphicx**. C'est un *package* très complet qui nous offre de nombreuses possibilités pour afficher des images. Sa commande principale, celle qui permet d'inclure une image dans un document, est la commande `\includegraphics`. Elle prend en paramètre le chemin du fichier contenant notre image. Voici donc un code insérant une image dans un document.

```
1 \documentclass[french]{article}
2 \usepackage{babel}
3 \usepackage[utf8]{inputenc}
4 \usepackage[T1]{fontenc}
5 \usepackage{graphicx}
6
7 \begin{document}
8   Et ici, \includegraphics{garfield} on a inséré l'image d'un
9   chat.
10 \end{document}
```

Ici l'image s'est placé en plein milieu du texte. C'est normal, nous avons laissé l'image dans le même paragraphe que celui du texte qui l'entoure, il est donc dans ce paragraphe. Si nous voulons l'avoir à part, il nous faut changer de paragraphe.

```
1 Et le même chat.
2
3 \includegraphics{garfield}
4
5 Le même chat, mais cette fois centré.
6 \begin{center}
7   \includegraphics{garfield}
8 \end{center}
```

?

Pourquoi notre image n'a-t-elle pas d'extension ?

L'extension n'est pas obligatoire, LaTeX va trouver tout seul l'image. Cependant, il peut y avoir

II. Compléter vos documents

des conflits dans le cas où plusieurs images ont le même nom mais des extensions différentes. En fait, le *package* `graphicx` a une liste ordonnée d'extensions possibles ; il cherchera d'abord le fichier avec la première extension, puis la deuxième, etc.



Le dossier courant est celui du fichier compilé. Ainsi, en utilisant `subfiles` avec un sous-fichier dans un dossier différent de celui du fichier principal, les chemins des images ne seront pas les mêmes selon que l'on veuille compiler le fichier principal ou le fichier secondaire.

Un exemple sera plus parlant. Supposons un projet avec cette arborescence, le fichier `img.png` étant affiché dans `chap1.tex`.

```
1 projet.tex
2 img/
3   image
4 chap/
5   chap1.tex
```

Lorsque l'on veut compiler `chap1.tex`, le chemin de l'image doit être `../img/image`, mais lorsque l'on veut compiler `projet.tex`, il doit être `img/image`. Pour ne pas avoir à faire ces remplacements, nous pouvons indiquer à LaTeX où chercher les images.

Ici, nous voyons comment modifier l'ordre des extensions et comment indiquer à LaTeX un dossier où chercher les images.

👁 Contenu masqué n°8

12.1.2. Modifier l'image

Bien sûr, ce *package* dispose d'une grande quantité d'options, nous permettant d'indiquer de manière très précise la manière dont on veut afficher l'image. Nous n'allons pas toutes les voir, mais seulement les plus usuelles.

Nous pouvons spécifier les dimensions que nous voulons pour l'image affichée. Les options `width` et `height` permettent d'imposer respectivement la largeur et la hauteur de l'image. Lorsque nous n'utilisons qu'une des deux options, LaTeX modifie l'autre longueur pour que l'image ne soit pas déformée (l'image est redimensionnée, mais les proportions sont les mêmes). Mais si nous spécifions les deux, l'image est bien évidemment déformée.

```
1 \includegraphics[height=5cm]{garfield}
2
3 \includegraphics[width=6cm]{garfield}
4
5 \includegraphics[width=6cm,height=5cm]{garfield}
```

Nous pouvons également utiliser l'option `scale` qui permet de redimensionner l'image en précisant un ratio. Un ratio de 1 signifie que l'image garde sa taille, un ratio de 2 que ses dimensions sont doublées, un ratio de 0.5 que ses dimensions sont diminuées de moitié. Bien sûr, ce ne sont pas les seuls ratios disponibles ; nous pouvons donner n'importe quel nombre. De

plus, si nous donnons un ratio négatif, l'image sera inversée.

```
1 \includegraphics[scale=0.5]{garfield}
2
3 \includegraphics[scale=1.5]{garfield}
4
5 \includegraphics[scale=-1]{garfield}
6
7 \includegraphics[scale=-0.5]{garfield}
```

12.1.3. Mode brouillon

Pour finir, il nous faut parler de l'option `draft` du *package*. Elle permet d'accélérer la compilation en n'incluant pas les images, mais en mettant des rectangles à la place. De plus, elle sert également si nous avons besoin d'aide. En effet, avec l'option `draft`, les images ne sont pas nécessaires et donc nous pouvons poster notre code et les personnes voulant nous aider n'auront pas besoin de l'image pour compiler notre code.

```
1 \usepackage[draft]{graphicx}
2
3 \begin{document}
4   \includegraphics[width=3cm]{garfield}
5 \end{document}
```



Si l'option `draft` est présente dans notre `documentclass`, nous n'avons pas besoin de la spécifier pour `graphicx`, il sera chargé avec cette option.

Notons que l'option `draft` peut être utilisée de manière locale en tant qu'option de la commande `\includegraphics`. Ainsi, l'image associée ne sera pas affichée ; elle passe en mode brouillon.

12.2. Tableaux

12.2.1. Composer un tableau

Les tableaux sont créés avec l'environnement `tabular`. Il prend en paramètre une description des colonnes du tableau suivant cette règle : on définit une nouvelle colonne à partir d'une lettre qui indique l'alignement du texte de cette colonne.

- `l` signifie « aligné à gauche » (*left*).
- `r` signifie « aligné à droite » (*right*).
- `c` signifie « centré » (*center*).

Avec `\begin{tabular}{rlc}`, on définit un tableau à trois colonnes, le texte de la première aligné à droite, celui de la deuxième à gauche, et celui de la dernière centré.

On définit ensuite le contenu du tableau (de gauche à droite et de haut en bas) dans l'environnement en suivant les règles qui suivent.

- On passe à la colonne suivante avec `&`.

II. Compléter vos documents

— On passe à la ligne suivante avec `\\`.
Ainsi, on peut faire ce code.

```
1 \begin{tabular}{lll}
2   1 - 1 & 1 - 2 & 1 - 3 \\
3   2 - 1 & 2 - 2 & 2 - 3 \\
4   3 - 1 & 3 - 2 & 3 - 3 \\
5 \end{tabular}
```



Nous devons bien entendu respecter le nombre de colonnes que nous avons spécifié avec l'argument de `tabular`. Ainsi, ce code nous donnera une erreur.

```
1 \begin{tabular}{lll}
2   1 & 2 & 3 & 4 \\
3 \end{tabular}
```

12.2.1.1. Tableaux et paragraphes

Notons qu'un tableau fait partie du paragraphe dans lequel il est. Ainsi, avec le code qui suit, le tableau est intégré au texte.

```
1 Un tableau
2 \begin{tabular}{c}
3   Une ligne\\
4   Une autre ligne
5 \end{tabular}
6 et sa suite.
```

En fait, l'environnement `tabular` prend en option l'alignement vertical du tableau : `c`, l'option par défaut pour le centrer, `b` (*bottom*) pour l'aligner en bas et `t` (*top*) pour l'aligner en haut. Nous pouvons alors écrire ces codes.

```
1 Un tableau
2 \begin{tabular}[b]{c}
3   Une ligne\\
4   Une autre ligne
5 \end{tabular}
6 et sa suite.
7
8 Un tableau
9 \begin{tabular}[t]{c}
10  Une ligne\\
11  Une autre ligne
12 \end{tabular}
```

II. Compléter vos documents

```
13 et sa suite.
```

Pour avoir le tableau à part, il nous faudra donc le placer dans son propre paragraphe.

```
1 Un tableau...
2
3 \begin{tabular}{c}
4   Une ligne\\
5   Une autre ligne
6 \end{tabular}
7
8 La suite.
```

Néanmoins, il est un peu collé au reste du texte. Pour le détacher, nous allons le placer dans un environnement d'alignement (donc `flushleft`, `center` ou `flushright`). Observons la différence avec ce code.

```
1 Un tableau...
2
3 \begin{flushleft}
4 \begin{tabular}{c}
5   Une ligne\\
6   Une autre ligne
7 \end{tabular}
8 \end{flushleft}
9
10 La suite.
```

12.2.1.2. Filets

Pour le moment, nos tableaux ne sont pas très beaux. Rajoutons des filets.

L'ajout de filets verticaux se fait dans l'argument de l'environnement `tabular` grâce au symbole `|`. Avec `{|l|c|}`, on rajoute des filets entre la première et la seconde colonne, et après la seconde colonne.

Les filets horizontaux, eux, sont définis avec le contenu du tableau grâce à la commande `\hline`. Il suffit de la placer avant le contenu d'une ligne pour avoir un filet horizontal avant cette ligne. On peut alors compléter notre tableau précédent de cette manière.

```
1 \begin{tabular}{|l|l|l|l|} \hline
2   1 - 1 & 1 - 2 & 1 - 3 \\ \hline
3   2 - 1 & 2 - 2 & 2 - 3 \\ \hline
4   3 - 1 & 3 - 2 & 3 - 3 \\ \hline
5 \end{tabular}
```

Qui nous donne le résultat suivant.

II. Compléter vos documents

1 - 1	1 - 2	1 - 3
2 - 1	2 - 2	2 - 3
3 - 1	3 - 2	3 - 3

Notons qu'il est possible d'avoir des doubles filets. Essayons par exemple ce code.

```
1 \begin{tabular}{|l|l|l|l|} \hline
2 1 - 1 & 1 - 2 & 1 - 3 \\ \hline
3 2 - 1 & 2 - 2 & 2 - 3 \\ \hline \hline
4 3 - 1 & 3 - 2 & 3 - 3 \\ \hline
5 \end{tabular}
```

12.2.1.2.1. Filet horizontal partiel Et, nous pouvons également placer des filets horizontaux partiels grâce à la commande `\cline`. Avec `\cline{i-j}`, on crée un filet horizontal entre les colonnes `i` et `j` du tableau. Voici un exemple de code utilisant `\cline`.

```
1 \begin{tabular}{|l|l|l|l|} \hline
2 1 - 1 & 1 - 2 & 1 - 3 \\ \cline{1-1} \cline{3-3}
3 2 - 1 & 2 - 2 & 2 - 3 \\ \hline
4 3 - 1 & 3 - 2 & 3 - 3 \\ \hline
5 \end{tabular}
```

1 - 1	1 - 2	1 - 3
2 - 1	2 - 2	2 - 3
3 - 1	3 - 2	3 - 3

Avec ce code, nous créons un tableau avec un deuxième filet horizontal incomplet (il n'est que sous la première et la troisième colonne).

12.2.1.3. Répéter un descripteur

Lorsque nous avons plusieurs fois de suite le même type de commande, nous ne sommes pas obligés de toutes les écrire. Il nous suffit de le remplacer par `*{<nombre>}{format}`. Ainsi, `{|c|*{3}{|l|}c|}` est équivalent à `{|c|l|l|l|c|}`. Bien sûr, lorsque nous avons peu de colonnes, cela n'est pas très utile, mais lorsqu'on se retrouve à devoir faire un tableau de 10 colonnes à l'alignement identique, ça peut être un plus.

12.2.2. Modifier le tableau

12.2.2.1. Fixer la largeur d'une colonne

Nous pouvons de plus fixer la largeur d'une colonne (par défaut, elles dépendent de la largeur du contenu). En fixant la taille d'une colonne, on obtient ce qu'on pourrait appeler une « colonne-

II. Compléter vos documents

paragraphe ». Cela se fait avec le descripteur `p{<taille>}`. Par exemple, essayons ce code.

```
1 \begin{tabular}{|l|p{2.5cm}|} \hline
2 1 - 1 & Du texte \\ \hline
3 2 - 1 & On met un grand texte qui sera sur plusieurs lignes \\
4 \end{tabular}
```

Le texte est alors justifié.

12.2.2.2. Choisir le séparateur de colonne

Pour le moment, nos colonnes étaient soit séparées par un filet vertical, soit par rien. Pourtant, il est possible de choisir un séparateur en écrivant `@{<symbole>}`. On peut par exemple choisir d'avoir \$ comme séparateur.

```
1 \begin{tabular}{|l@{\$}l|l|} \hline
2 1 - 1 & 1 - 2 & 1 - 3 \\ \hline
3 2 - 1 & 2 - 2 & 2 - 3 \\ \hline
4 3 - 1 & 3 - 2 & 3 - 3 \\ \hline
5 \end{tabular}
```

12.2.2.3. Notes de bas de page dans un tableau

Essayons de faire une référence dans un tableau.

```
1 \begin{tabular}{|l|l|} \hline
2 Ligne 1 \footnote{et colonne 1} & Ligne 2 \\ \hline
3 \end{tabular}
```



Et ce code ne fonctionne pas.

Pour faire une note de bas de page dans un tableau, il nous faut utiliser `\footnotemark` (nous avons enfin un exemple d'un cas où `\footnote` ne répond pas à nos besoins). Ce qui nous donne ce code.

```
1 \begin{tabular}{|l|l|} \hline
2 Ligne 1 \footnotemark & Ligne 2 \\ \hline
3 \end{tabular}
4 \footnotetext{et colonne 1}
```

12.2.2.4. Fusion de cellule

12.2.2.4.1. Fusion de ligne Nous avons déjà fait une fusion de lignes un peu plus haut lorsque nous avons utilisé `\cline`. Ne pas mettre de filet horizontal entre deux cellules de deux lignes

II. Compléter vos documents

permet de fusionner ces deux cellules.

i

Pour fusionner des lignes plus simplement, nous pourrions nous tourner vers le *package* `multirow`.

12.2.2.4.2. Fusion de colonne La fusion de deux colonnes, elle, se fait à l'aide de la commande `\multicolumn`. Elle prend trois paramètres, le nombre de colonnes qui doivent être fusionnées, le descripteur que l'on veut pour cette nouvelle colonne, et son contenu.

```
1 \begin{tabular}{|l|l|l|} \hline
2   1 - 1 & 1 - 2 & 1 - 3 \\ \hline
3   2 - 1 & \multicolumn{2}{c}{2 à 3} \\ \hline
4   3 - 1 & 3 - 2 & 3 - 3 \\ \hline
5 \end{tabular}
```

Ici par exemple, nous remplaçons les deux dernières colonnes de la deuxième ligne par une colonne où le contenu, « 2 à 3 », sera centré. On obtient donc ce résultat.

1 - 1	1 - 2	1 - 3
2 - 1	2 à 3	
3 - 1	3 - 2	3 - 3

12.2.2.4.3. Supprimer la première cellule d'un tableau En utilisant conjointement `\cline` et `\multicolumn`, on arrive à supprimer la première cellule d'un tableau. Il suffit de ne pas tracer de filet horizontal au dessus de cette cellule (grâce à `\cline`) et d'utiliser `\multicolumn` sur cette cellule pour ne pas placer de filet vertical à sa gauche.

```
1 \begin{tabular}{|l|l|} \cline{2-2}
2   \multicolumn{1}{c}{ } & 1 \\ \hline
3   2 & 3 \\ \hline
4 \end{tabular}
```

L'utilisation conjointe de toutes ces fonctions permet alors de faire des tableaux plus complexes, comme celui qui suit.

```
1 \newcommand{\chef}[1]{\textsc{\textbf{#1}}}
2
3 \begin{tabular}{|l|c|c|}
4   \begin{longtabu}{|p{\dimexpr\linewidth /
5     3|p{\dimexpr\linewidth / 3|p{\dimexpr\linewidth / 3|}
6   \hline
```

```

4 \rowfont[c]{\bfseries}
5 1 - 1 & \multirow{2}{*}{1 - 2 \par \par 2 - 2} & 1 - 3 \\
6 \cline{1-1}
7 \rowfont[l]{}
8 2 - 1 & & 2 - 3 \\ \hline
9 3 - 1 & 3 - 2 & 3 - 3 \\ \hline
10 \end{longtabu}
11 \cline{2-3}
12 \multicolumn{1}{c|}{} & \multicolumn{2}{c|}{Informations}
13 \multicolumn{1}{c|}{} & Langue officielle & Chef du
14 gouvernement \\ \hline
15 France & Français & \chef{François}
16 HOLLANDE} \\ \hline
17 Royaume-Uni & Anglais & \chef{Theresa}
18 MAY} \footnotemark \\ \hline
19 États-Unis d'Amérique & Anglais & \chef{Barack}
20 OBAMA} \\ \hline
21 \end{tabular}
22 \footnotetext{Au Royaume-Uni, le chef du gouvernement est le
23 \emph{Premier ministre}.}

```



De nombreux *packages* [↗](#) étendent nos possibilités en termes de composition de tableaux. Du simple **array**, au beaucoup plus complet **booktabs** ou encore **tabu**. Il y en a plein, et c'est plutôt dur de se repérer à ce sujet. Là encore, il ne faut pas hésiter à regarder les différents avis sur ces *packages*, voir lesquels fonctionnent ensemble, etc.

12.3. Texte préformaté

Un texte préformaté est un texte qui doit apparaître tel qu'il a été saisi, c'est-à-dire que rien ne sera interprété. Les commandes seront affichées, tous les espaces seront affichés, etc. Il peut être utile pour présenter du code par exemple. Traditionnellement, le texte préformaté est dans une police à chasse fixe.

Pour écrire du texte préformaté avec LaTeX, nous pouvons utiliser l'environnement `\verbatim`.

```

1 \begin{verbatim}
2 Dans l'environnement verbatim, rien n'est interprété. On peut
3 même écrire
4 n'importe quelle commande sans problème.
5 \begin
6 \end{verbatim}

```

En compilant ce code, on obtient bien le résultat voulu.

12.3.1. La commande `\verb`

Si on veut placer du texte préformaté dans du texte, on ne peut pas utiliser l'environnement `verbatim`. On va utiliser la commande `\verb`. Elle prend en paramètre le texte que l'on veut afficher, mais le paramètre n'est pas entre accolades, mais est entouré d'un même caractère. On peut choisir n'importe quel caractère qui n'apparaît pas dans le texte passé en paramètre.

```
1 | En LaTeX, on obtient une section avec la commande \verb|section|.
```

Notons que la commande `\verb` ne peut pas être utilisée dans l'argument d'une commande. Par exemple, écrire `\section{\verb|S|}` nous donnera l'erreur « LaTeX Error : `\verb` illegal in command argument ».

i

Pour bénéficier de plus de fonctionnalités (dont la coloration syntaxique), nous pourrions nous renseigner sur les *packages* qui permettent l'affichage de code dans un document [↗](#). Notons particulièrement les packages **listings** et **minted**, le dernier étant présenté dans ce [tutoriel](#) [↗](#). Pour afficher des algorithmes, nous pourrions par exemple regarder le *package* **algorithmic**.

Avec ce chapitre, nous avons de quoi faire des documents beaucoup plus complets. Mais nous n'en avons pas encore fini avec la gestion de ces différents éléments. En effet, en LaTeX, on parle souvent de flottants et de figures, et c'est un aspect qu'il nous reste à aborder... Dans un prochain chapitre.

Contenu masqué

Contenu masqué n°8

L'ordre des extensions peut être modifié en utilisant la commande `\DeclareGraphicsExtension` qui permet de choisir une liste d'extensions.

Notons de plus l'existence de la fonction `\graphicspath` qui prend en paramètre des dossiers dans lesquels chercher les images. Si on a par exemple beaucoup d'images dans le dossier `rsc/png`, ça peut être une bonne idée d'utiliser `\graphicspath`.

```
1 | \graphicspath{{rsc/}{img/}}
2 | \DeclareGraphicsExtension{png, pdf, jpg}
3 | \includegraphicx{piano}
```

Ici, les images seront d'abord cherchées dans le dossier courant, puis dans le dossier `rsc`, et enfin dans le dossier `img` et on recherchera d'abord le fichier `piano.png`, puis le fichier `piano.pdf` et enfin le fichier `piano.jpg`.

Nous pouvons alors régler le problème que nous avons avec subfiles. En effet, il nous suffit par exemple d'utiliser `\graphicspath` dans le fichier `chap1.tex` de notre exemple pour indiquer à LaTeX de chercher dans `../img`.

II. Compléter vos documents

```
1 \documentclass[../projet.tex]{subfiles}
2 \graphicspath{{../}}
3
4 \begin{document}
5     \includegraphics{img/image}
6 \end{document}
```

Ici, si on compile `projet.tex`, LaTeX cherchera `img/image` et le trouvera; si on compile `chap1.tex`, LaTeX cherchera `img/image`, ne trouvera pas, cherchera `../img/image` et trouvera.

En clair, une stratégie simple est d'utiliser `\graphicspath` dans les sous-fichiers en donnant pour chaque dossier deux chemins, celui depuis le dossier du fichier principal, et celui depuis celui du sous-fichier.

[Retourner au texte.](#)

13. Les flottants

Après avoir appris à rajouter des images et des tableaux dans notre document, il est temps de voir la notion de flottants. Ils nous permettent de laisser LaTeX gérer la position des images et des tableaux dans notre document.

i

Les codes de ce chapitre contiennent des inclusions d'images. Pour les tester, le plus simple est certainement d'utiliser l'option `draft`.

13.1. Les flottants

13.1.1. Qu'est-ce qu'un flottant ?

Quand on se renseigne sur LaTeX et l'inclusion d'images et de figures, on ne peut pas passer à côté des flottants. On lit parfois même que pour inclure une image dans un document, il nous faut utiliser un flottant (c'est faux, nous n'avons pas encore utilisé de flottants et pourtant nous avons vu comment placer des images dans un document).

Comme son nom l'indique, un flottant, c'est grossièrement un truc qui flotte. Pour être un peu plus précis et indiquer ce qu'on veut dire par « flotter », il nous faut rappeler qu'avec LaTeX, nous ne choisissons pas l'emplacement exact des objets sur la page, nous indiquons la sémantique du document (par exemple, il doit y avoir cette image après tel paragraphe).

Le problème qui vient alors est qu'une image peut alors être placée d'une manière qu'on va juste qualifier de moche, laissant par exemple un quart de page vide pour être placée à la page suivante.

En utilisant des flottants, on demande à LaTeX de placer les image « au mieux ». En fait, en utilisant un flottant, on lui indique que s'il ne peut pas la placer exactement à l'endroit voulu et obtenir un résultat esthétique (pour des critères d'esthétismes définis par LaTeX), alors il peut essayer de la placer quelque part d'autre, au plus près.

13.1.2. Utiliser les flottants

Les flottants se présentent sous la forme d'environnements. Par défaut, LaTeX a deux environnements pour les flottants.

- L'environnement `figure` pour faire flotter les images et autres dessins.
- L'environnement `table` pour faire flotter les tableaux.

Leur utilisation est très simple, le contenu de l'environnement est le contenu à faire flotter. Voici un code utilisant les deux concepts.

```
1 \begin{table}
2   \centering
```

```

3  \begin{tabular}{|l|c|c|}
   \cline{2-3}
4  \multicolumn{1}{c|}{} & \multicolumn{2}{c|}{Informations}
   \\ \cline{2-3}
5  \multicolumn{1}{c|}{} & Langue officielle & Chef du
   gouvernement
6  France & Français & \bsc{François
   HOLLANDE} \\ \cline{2-3}
7  Royaume-Uni & Anglais & \bsc{Theresa
   MAY} \footnotemark \\ \cline{2-3}
8  États-Unis d'Amérique & Anglais & \bsc{Barack
   OBAMA} \\ \cline{2-3}
9  \end{tabular}
10 \footnotetext{Au Royaume-Uni, le chef du gouvernement est le
   \emph{Premier ministre}.}
11 \end{table}
12
13 Un peu de texte
14
15 \begin{figure}
16   \centering
17   \includegraphics[width=10cm, height=7cm]{image}
18 \end{figure}

```

i

Généralement, pour centrer un flottant on n'utilise pas l'environnement `center`, mais juste la commande `\centering` dans l'environnement utilisé pour le flottant.

Le contenu de l'environnement n'est pas « vérifié » par LaTeX ; nous pourrions tout à fait mettre du texte dans un environnement `figure` ou une image dans un environnement `table`. Mais bon, tâchons de respecter la sémantique de ces environnements. Si par exemple nous voulons inclure un tableau et que nous disposons de celui-ci en image, ce ne serait pas une erreur d'utiliser `table`.

13.1.3. Légendes et références

Avec un flottant, il est possible de donner une légende à nos éléments. Ceci se fait simplement en utilisant la commande `\caption` dans l'environnement flottant. Elle prend en argument la légende que l'on veut donner.

Les flottants sont numérotés et l'affichage suit alors la forme « Figure 2 - légende » par exemple s'il s'agit de la deuxième figure et « Table 2 - légende » s'il s'agit de la deuxième table. Ceci étant, nous pouvons afficher la liste des figures, en utilisant la commande `\listoffigures` et celles des tables avec `\listoftables`.

LaTeX a donc un compteur pour les figures et un compteur pour les tables, et nous pouvons faire donc référence à ces éléments en utilisant `\ref`, le `\label` étant à l'intérieur du flottant, après le `\caption`.

!

N'oublions pas de compiler deux fois pour prendre en compte les références.

```
1 \begin{figure}
2   \centering
3   \includegraphics[width=10cm, height=7cm]{image1}
4   \caption{Image 1}
5   \label{fig:image1}
6 \end{figure}
7
8 \begin{figure}
9   \centering
10  \includegraphics[width=10cm, height=7cm]{image2}
11  \label{fig:image2}
12 \end{figure}
13
14 \begin{figure}
15   \centering
16   \includegraphics[width=10cm, height=7cm]{image3}
17   \caption{Image 3}
18   \label{fig:image3}
19 \end{figure}
20
21 On a une première image à la figure \ref{fig:image1} et une autre
22 à la figure \ref{fig:image3}
23
24 \listoffigures
```

Nous remarquons que la numérotation a sauté la figure pour la deuxième 2 ; ce qui est normal vu que l'on ne lui a pas donné de légende. Cela signifie notamment qu'on ne pourra pas faire référence à ce flottant (et donc son `\label` est inutile). D'ailleurs, la deuxième figure n'est pas référencée dans la liste des figures.

13.2. Placer les flottants

13.2.1. Les options de placement

LaTeX ne place pas forcément les flottants à leur position dans le code source, mais où les place-t-il donc ? En fait, les environnements des flottants ont une option qui correspond à nos souhaits de placement. Il y en a quatre.

- `h` pour *here* demande à placer le flottant là où il apparaît dans le code source.
- `t` pour *top* demande à le placer en haut de la page.
- `b` pour *bottom* demande à le placer en bas de la page.
- `p` pour *page* demande à le placer sur une page à part, qui ne contiendra pas de texte.

Nous pouvons faire plusieurs souhaits, en mettant les lettres de nos souhaits dans l'option. Par exemple avec `\begin{figure}[htp]` on fait trois demandes.

L'ordre de nos souhaits n'a pas d'importance, LaTeX utilise toujours le même ordre qui est `htbp`. Ainsi, les demandes `hbt` ou `htb` ou encore `thb` sont équivalentes. LaTeX essaiera d'abord de placer l'image à sa place dans le code source, puis essaiera de la placer en haut de page, et enfin en bas de page.

Si aucune option n'est précisé, LaTeX utilise `tbp`, c'est-à-dire qu'il ne cherche même pas à placer

le flottant à sa place dans le texte. Si on veut qu'il essaie de le faire, il faudra donc forcément utiliser l'option.



LaTeX ne considère que les options données par l'utilisateur (ou celle par défaut si l'utilisateur n'en a pas donnée). Cela signifie que si on fournit `h` seulement en option, LaTeX ne cherchera à placer le flottant qu'à sa place dans le texte. Et s'il n'y arrive pas, il n'essaiera pas de le placer autre part.

En fait, dans le cas particulier de `h`, nous obtenons un avertissement lors de la compilation qui nous informe que notre `h` a été changé en `ht` si LaTeX ne réussit pas à placer la figure à son emplacement dans le code source.

Néanmoins, il vaut mieux toujours faire au moins deux demandes. Plus il y en a, plus LaTeX sera en mesure de placer notre flottant efficacement.

Notons de plus ces petites règles de placement :

- les flottants d'un même type apparaissent dans le document dans le même ordre que dans le fichier source. Cela signifie qu'une figure A qui apparaît après une figure B dans notre code source ne pourra pas être placée avant elle dans le document, et ce même si elle est beaucoup plus petite et qu'elle aurait pu être placée. Il n'y a par contre aucune contrainte d'ordre sur les flottants de types différents.
- Un flottant ne peut pas se trouver sur une page antérieure à celle de son placement dans le code source. S'il apparaît son placement dans le code source, c'est forcément sur la même page (par exemple en haut de la page). Il peut néanmoins se retrouver sur n'importe laquelle des pages suivantes.

13.2.2. Des légendes sans flottants

Il peut nous arriver de vouloir référencer ou légender un flottant, mais sans que celle-ci ne flotte. Pour cela, il y a plusieurs solutions.

Le plus simple est certainement d'utiliser le *package* **capt-of** qui offre la commande `\captionof`. Elle prend en paramètre le type de flottant associés (`figure` ou `table`), et la légende que l'on veut. Cette commande n'a pas besoin de flottant pour être utilisée. Il nous suffit donc de l'utiliser **sans utiliser de flottant**.

```
1 On a les images suivantes.
2
3 Pour commencer, une image de Garfield.
4
5 \begin{center}
6   \includegraphics[width=10cm, height=10cm]{garfield}
7   \captionof{figure}{En fait, les chats ne flottent pas ?}
8   \label{fig:garfield}
9 \end{center}
10
11 Puis une image de Milou.
12
13 \begin{center}
14   \includegraphics[width=10cm, height=10cm]{milou}
15   \captionof{figure}{Les chiens ça flotte pas non plus !}
```

```
16 \label{fig:milou}
17 \end{center}
18
19 Et enfin une image de Dumbo, l'éléphant volant.
20
21 \begin{center}
22 \includegraphics[width=10cm, height=10cm]{dumbo}
23 \captionof{figure}{Dumbo c'est une exception chez les éléphants
24 !}
25 \label{fig:dumbo}
26 \end{center}
27
28 On a donc Garfield en figure \ref{fig:garfield}, Milou en figure
29 \ref{fig:milou} et Dumbo en figure \ref{fig:dumbo}.
```

Nous observons bien que les images sont placées exactement là où elles sont dans nos sources et ce même si cela laisse un grand blanc dans le PDF.

C'est fini pour ce chapitre, nous savons maintenant comment faire flotter certains éléments. Dans beaucoup de situations, c'est une bonne idée d'utiliser des flottants et de laisser LaTeX placer les éléments au mieux. De nombreux *packages* améliorent la gestion des flottants et donnent de nouvelles fonctionnalités.

- Le *package* **caption** complète le *package* **capt-of** et permet de gérer les légendes de manières approfondies.
- Le *package* **float** permet de gérer le style des flottants, offre une interface simple pour en créer de nouveaux types et de manière générale améliore leur gestion.
- Le *package* **wrapfig** permet d'intégrer une image à du texte (le texte enveloppera l'image).
- Le *package* **subcaption** permet de placer des sous-flottants dans un flottant.



À venir

Une nouvelle section viendra compléter ce chapitre et expliquera comment LaTeX place les flottants dans le document (et comment modifier ce fonctionnement).

Conclusion

i

Ce tutoriel n'est pas fini. D'autres chapitre viendront bientôt compléter ceux qui sont là.

[[information]] | Ce tutoriel n'est pas fini. D'autres chapitre viendront bientôt compléter ceux qui sont là.

Pour poursuivre l'apprentissage, vous pouvez [regarder les autres contenus LaTeX de Zeste de Savoir](#) [↗](#), aller vous promener sur le site du CTAN, regarder du côté de XeLaTeX et de LuaLaTeX, etc.

De gros remerciements à @Saroupille, à @ct pour leurs remarques, à @TD qui a autorisé l'utilisation de certaines parties de son tutoriel et à @charlie02 qui a relevé des erreurs orthotypographiques. Et surtout, merci à @Gabbro qui a validé ce tutoriel et aux lecteurs.

Liste des abréviations

CTAN Comprehensive TeX Archive Network. 3, 98, 99

HTML HyperText Markup Language. 11–13

WYSIWYG What You See Is What You Get (ce que vous voyez est ce que vous obtenez). 1, 11, 12

WYSIWYM What You See Is What You Mean (ce que vous voyez est ce que vous voulez dire). 1, 11, 12

XML Extensible Markup Language. 11–13