Semestre Rebondir: **Informatique**

Irène Durand

11 mai 2009

Table des matières

Chapitre 1

Intéraction avec le système

1.1 L'interprète de commandes

On peut interagir avec le système
- soit de manière graphique (en cliquant sur des icônes ou en sélectionnant des choix dans des menus),
- soit en tapant les commandes équivalentes dans une fenêtre `terminal`.

1.1.1 Principes de base

Le programme qui s'exécute dans une fenêtre terminal s'appelle un `shell` ou `interprète de commandes`. Le shell affiche une `invite` (`prompt`) dans l'attente d'une commande tapée par l'utilisateur. Il y a plusieurs shells possible ; nous utiliserons le shell installé pas défaut : `Bash`.

Quand après avoir tapé une commande, l'utilisateur appuie sur la touche `entrée`, le shell analyse la commande, l'exécute puis *rend la main à l'utilisateur* en affichant à nouveau le prompt.

Une commande est une suite de mots, le premier étant le nom de la commande et les mots suivants les arguments fournis. Dans l'exemple suivant le prompt est matérialisé par le caractère `$`. `date, uname, echo, history` sont des noms de commandes.

Exemples de commandes

```
$ date
jeu jui 10 12:35:01 CEST 2008
$ uname
Linux
$ echo cette commande affiche ses arguments
cette commande affiche ses arguments
$ history
  ...
  704  date
  705  uname
  706  history
```

1.1.2 Édition d'une ligne de commande

Le shell offre des possibilités de déplacement et d'édition au niveau de la ligne. Ces fonctionnalités sont accessibles par l'intermédiaire de *clés* (touches ou combinaisons de touches).

Insertion

La frappe d'un caractère normal insère le caractère juste à avant le curseur et avance ce dernier d'une position.

Validation/Annulation

`Return`	Validation
`C-j`	Validation
`C-c`	Annulation

Déplacements

`C-a`	début de ligne
`C-e`	fin de ligne
`C-b` ou (←)	caractère précédent
`C-f` ou (→)	caractère suivant
`M-b`	début de mot
`M-f`	fin de mot

Si le clavier ne dispose pas de touche `M` (META), on utilise la touche `Alt` ou la touche `Esc`.

Suppressions

`C-d`	effacement caractère sous le curseur
`DEL`	effacement du caractère précédent
`C-k`	effacement du curseur à la fin de la ligne
`M-d`	effacement du mot suivant le curseur
`M-DEL`	effacement du mot précédent le curseur

Historique

La commande `history` montre l'historique des commandes. On peut naviguer dans l'historique des commandes et réutiliser tout ou partie d'une commande qui s'y trouve.

`C-p` ou (↑) rappelle la dernière commande
`C-n` ou (↓) rappelle la commande suivante (si on n'est pas sur la dernière)
`C-r` recherche une commande parmi les commandes précédentes

1.2 Configuration de de l'interprète `Bash`

`Bash` possède des variables dont les valeurs paramètrent l'environnement de travail. En changeant la valeur des variables, on peut changer le comportement du système. La commande `env` affiche la liste de ces variables. Pour accéder à la valeur d'une variable nommée `VAR` il faut écrire `$VAR`.

Les variables du shell

```
$ env
NNTPSERVER=news.u-bordeaux.fr
HOSTNAME=chataigne.labri.fr
SHELL=/bin/bash
TERM=xterm
USER=idurand
USERNAME=idurand
PATH=/usr/kerberos/bin:/usr/lib/ccache:/usr/local/bin:/usr/bin:/bin:/usr/labri/idurand/bin:/opt/pro
g/CMUCL/bin:/opt/prog/jdk/bin:/opt/prog/ocaml/bin:.:/usr/labri/idurand/bin:/opt/prog/CMUCL/bin:/opt
/prog/jdk/bin:/opt/prog/ocaml/bin:.
PWD=/usr/labri/idurand/Enseignement/Rebondir/COURS-REBONDIR
EDITOR=emacs
LANG=fr_FR.UTF-8
HOME=/usr/labri/idurand
LOGNAME=idurand
PRINTER=caxton
OLDPWD=/usr/labri/idurand/Enseignement/Rebondir/COURS-REBONDIR/JavaScript
...
```

Par exemple, on peut changer la valeur de la variable `PRINTER` pour imprimer sur une autre imprimante.

La variable `LANG` indique la langue par défaut utilisée par le système (`fr_FR` français de France) et le type d'encodage des caractères (`UTF-8`).

Subsitution de variable

Si un des arguments d'une commande est le nom d'une variable précédé du caractère `$`, il est remplacé par la valeur de la variable avant exécution de la commande proprement dite. Le mécanisme de remplacement d'une variable par sa valeur s'appelle *substitution de variable*.

La commande `echo` affiche ses arguments.

```
$ echo Imprimante: $PRINTER
Imprimante: caxton
```

Affectation d'une valeur à une variable et accès à la valeur

```
$ PRINTER=basquiat
$ echo $PRINTER
basquiat
$ ANNEE=2009
$ MOIS=02
$ echo cal $MOIS $ANNEE
cal 02 2009
$ cal $MOIS $ANNEE
    février 2009
di lu ma me je ve sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

Pour qu'une affectation soit valable dans tous les shells il faut la faire précéder par `export` :

```
$ export EDITOR=emacs
```

Alias

Un *alias* permet de donner un "petit nom" à une commande. La syntaxe est : `alias nom=<commande>` Si la commande contient plusieurs mots, il faut les entourer de quotes simples (`'`).

```
$ alias f=finger
$ alias ll='ls -l'
$ alias rm='rm -i'
```

La commande `alias` sans paramètre liste les `alias` existant.

```
$ alias
alias f=finger
alias ll='ls -l'
alias rm='rm -i'
```

Aide en ligne

La commande `man` (pour manuel) permet d'obtenir l'aide en ligne de la plupart des commandes.

Paramétrage pérenne

On peut paramétrer de manière permanente son environnement (son shell) en mettant des commandes shell dans son fichier `.bash_profile`. Les commandes contenues dans ce fichier sont automatiquement exécutées au lancement de `Bash`.

1.3 Exercices

Exercice 1

- Essayer les commandes suivantes `who, tty, date, cal, echo, banner`.
- Commenter leur fonctionnement.

Exercice 2

- Essayer `man banner, man cal, man tty`.
- Essayer d'obtenir de l'aide en ligne sur le manuel ?

Exercice 3

- Rechercher une commande qui efface le contenu de la fenêtre `terminal`.
- Utiliser la commande `apropos` avec des mots-clés (*e.g.* `screen`, `terminal`).

Exercice 4

- Essayer les commandes d'édition de ligne.
- faire afficher le calendrier de l'année courante ;

– utiliser l'historique et l'édition de ligne pour afficher successivement les calendriers des mois de janvier, de décembre, et de février de l'année courante. Essayer la recherche avec `C-r`.

Exercice 5

– Sélectionner un texte dans une fenêtre `terminal` pour ensuite le coller dans une autre fenêtre `terminal`.
– Vérifier la possibilité de faire la même opération de copier-coller entre du texte contenu dans votre navigateur Web et une fenêtre `terminal`.

Exercice 6

– Examiner les variables présentes dans votre environnement. Affecter une valeur à une nouvelle variable.
– Faire afficher la valeur de la variable.

Exercice 7

– Afficher la liste des `alias` définis dans le shell courant.
– Créer et tester des alias.
– Définir un alias `date` sur la commande `ls`.
– Détruire l'alias précédent et rappeler la commande `date` afin d'en vérifier son bon fonctionnement.

Chapitre 2

Fichiers et répertoires

2.1 Arborescence et chemins

Le système est constitué d'un ensemble de *fichiers* organisés en répertoires. L'ensemble forme une arborescence (en fait un dag ou graphe orienté acyclique).

On accède à un noeud de l'arborescence (répertoire ou fichier) par un *chemin* constitué par la séquence des noms de répertoires suivis par le caractère `/`.

La racine de l'arborescence est symbolisée par le caractère `/`.

Une chemin peut être *absolu*, c'est-à-dire indiqué depuis la racine (et donc commençant par un `/`) ou *relatif*, c'est-à-dire indiqué à partir du répertoire dans lequel on se trouve.

```
$ pwd
/net/cremi/irdurand
```

2.2 Parcours de l'arborescence

2.2.1 `ls`

La commande `ls` affiche le contenu du répertoire courant. Avec un argument de type chemin (absolu ou relatif), elle affiche le contenu du répertoire si le chemin désigne un répertoire.

```
$ ls
$ ls ~
$ ls /usr/lib
$ ls Rebondir/Informatique
```

La commande `ls` a de multiples options (voir `man ls`. L'option `-a` permet de voir les fichiers et répertoires *cachés* dont le nom commence par un `.`. Ces fichiers et répertoires sont le plus souvent des fichiers de configuration ou d'historique de divers logiciels.

Le fichier `.bash_profile` permet de configurer le shell `Bash`par exemple de même que le fichier `.emacs` permet de configurer l'éditeur de textes `Emacs`.

Dans tout répertoire, il existe deux *entrées* particulières : `.` et `..`. On ne les voit donc que quand on utilise la commande `ls` avec l'option `-a`.

L'entrée `.` représente le répertoire lui-même ; ainsi `ls .` est équivalent à `ls`. L'entrée `..` représente le répertoire *père* (situé juste au-dessus dans l'arborescence). Par exemple, la commande `ls ..` permet de lister le contenu du répertoire parent.

2.2.2 cd

La commande `cd` (change-directory) permet de se déplacer dans l'arborescence. Sans argument, elle permet de se positionner dans son répertoire d'accueil. Avec un argument de type chemin (absolu ou relatif), elle permet de se positionner dans le répertoire correspondant au chemin.

```
$ cd /
$ pwd
$ cd net/cremi
$ cd irdurand/Musique
$ cd ../Desktop
$ pwd
```

2.3 Création/suppression d'un répertoire, d'un fichier

2.3.1 mkdir

La commande `mkdir` permet de créer un nouveau répertoire vide.

```
$ cd
$ mkdir Rebondir
$ mkdir Rebondir/Informatique
$ mkdir Rebondir/Physique
$ cd Rebondir
$ cd Physique
$ cd ../Informatique
```

2.3.2 rmdir

La commande `rmdir` permet de détruire un répertoire vide.

```
$ cd
$ mkdir Rebondir/Maths
$ rmdir Rebondir/Maths
```

2.3.3 touch

La commande `touch` appelée avec un (ou plusieurs) nom de fichier met la date de dernière modification du fichier à la date courante. Si le fichier n'existe pas, elle crée un fichier vide.

```
$ touch f1.txt
$ ls -l *.txt
$ toucht f1.txt f2.txt f3.txt
$ ls -l *.txt
```

2.3.4 emacs

La plupart du temps, on utilise un logiciel de type *éditeur de textes* pour créer et éditer des fichiers textes. Nous utiliserons `Emacs`(voir chapitre suivant).

Complétion

La clé `C-i` ou encore la touche TAB sert à compléter automatiquement les mots de la commande avec des noms de fichiers ou répertoire accessibles sur le système.

Par exemple `ty C-i` est complété en `type`. `cd D C-i C-i` propose plusieurs complétion si on continue avec `o C-i` on obtient la commande `cd Documents`.

2.4 Substitution de chemin

Dans les chemins, certains caractères ont une signification particulière. Le mécanisme de remplacement de ces caractères spéciaux par le shell s'appelle le mécanisme de *substitution de chemin*.

```
$ echo ~
$ echo *
$ ls D*
$ ls *.js
$ cd Documents
$ ls -ld .
$ cd ..
```

2.5 Mécanismes de quotation

Il existe des mécanismes de *quotation* pour rendre un caractère spécial *normal*

```
$ echo ~
$ /net/cremi/irdurand
$ echo \~
$ ~
```

Les caractères suivants sont spéciaux pour `Bash` : `~ * ? " . ‘ \ $`

2.6 Copies, déplacements, liens

2.6.1 `rm`

Cette commande permet de supprimer un fichier.

2.6.2 `mv`

Cette commande permet de renommer un fichier ou un répertoire, ou de déplacer plusieurs fichiers et répertoires dans un répertoire.

```
$ mv f1.txt f1-bis.txt # renommage simple
$ cd Rebondir/Informatique
$ touch f1.txt
$ mv f1.txt ../Physique/f1-bis.txt # renommage et déplacement
$ touch f2.txt f3.txt f4.txt
$ mv f2.txt ../Physique # déplacement d'un fichier (sans renommage)
$ mv f3.txt f4.txt  # déplacement de plusieurs fichiers (sans renommage)
$ mv Maths Mathematique # renommage de répertoire
```

2.6.3 `cp`

Cette commande permet de copier un fichier vers un autre répertoire (avec ou sans renommage) et de copier plusieurs fichiers vers un même répertoire (sans renommage).

```
$ cp f1.txt f1-bis.txt
$ cp f1.txt ../f1-bis.txt
$ cp f1.txt f2.txt ../Informatique
```

Copier un fichier dans le même répertoire n'est utile que si on renomme le fichier.

Grâce à l'option `-r` (récursivement) de la commande `cp`, on peut copier tout un répertoire.

```
$ cp -r Rebondir Rebondir-bis # copie d'un sous-arbre
```

2.6.4 `ln`

Cette commande permet de faire un lien vers un fichier ou un répertoire. Si le lien est fait dans le même répertoire le nom du lien doit être différent du nom du fichier ou répertoire.

2.6.5 Option `-i`

Les trois commandes vues précédemment (`mv, cp, ln`) admmettent l'option `-i` (mode interactif). Dans ce cas, en cas d'écrasement possible un fichier, la commande demande confirmation de cet écrasement auprès de l'utilisateur.

```
$ touch f1.txt
$ touch f2.txt
$ mv f1.txt f2.txt
mv: écraser ‘f2.txt’?n
$
```

2.7 Examen de fichiers texte

`cat, more, less`

2.8 Exercices

Exercice 1

Essayer

- `$ ls`
- `$ pwd`
- `$ cd ..`
- `$ pwd`
- `$ ls`

Exercice 2

Essayer les séquences suivantes

- `$ cd`
- `$ mkdir Rebondir`
- `$ ls R C-i`
- `$ cd R C-i`
- `$ ls`
- `$ pwd`

Exercice 3

Exécuter la suite de commandes :

- `$ cd Rebondir` (n'oubliez pas le mécanisme de complétion)
- `$ pwd`

Maintenant pour retourner dans son répertoire d'accueil, plusieurs possibilités :

- `$ cd ..` ou
- `$ cd` $\sim$ ou encore
- `$ cd`

Exercice 4

Dessiner son arborescence.

Exercice 5
Essayer différentes options de la commande `ls` dans votre répertoire d'accueil. Commenter. Y-a-t'il une notion de fichier caché ?

Exercice 6
Exécuter, en utilisant les mécanismes d'édition de[1] *bash*, les commandes

```
$ man ls
$ ls Desktop
$ ls -F Desktop
$ ls -a Desktop
$ ls -aF Desktop
$ ls -l Desktop
```

Exercice 7
Essayer `cat` $\prec fichier \succ$.

Exercice 8
Essayer `more` $\prec fichier \succ$. De même avec la commande `less`. Vaut-il mieux utiliser la commande `more` ou `less` et pourquoi ?

Exercice 9
Testez la commande `file`. Essayer

```
$ cd Desktop
$ file Trash
$ file *
$ file .*
```

(On peut se limiter pour l'instant à une utilisation simple du métacaractère `*`)

Exercice 10
Essayer

```
$ wc Trash
$ wc *
$ wc .*
```

$ `wc` : par défaut, lecture au clavier ; entrer des lignes et terminer en tapant `C-d` en début de ligne

Exercice 11
Essayer `echo ~`. Comment peut-on faire afficher le caractère `~` ? Commenter rapidement le mécanisme de **quotation** au moyen du caractère \.

Exercice 12
Essayer successivement :

```
$ touch '*'
$ ls
$ rm \*
$ ls
```

Exercice 13
Essayer `rmdir ~` et commenter le résultat obtenu.

[1] On dit **de** bash et nom *du* bash.

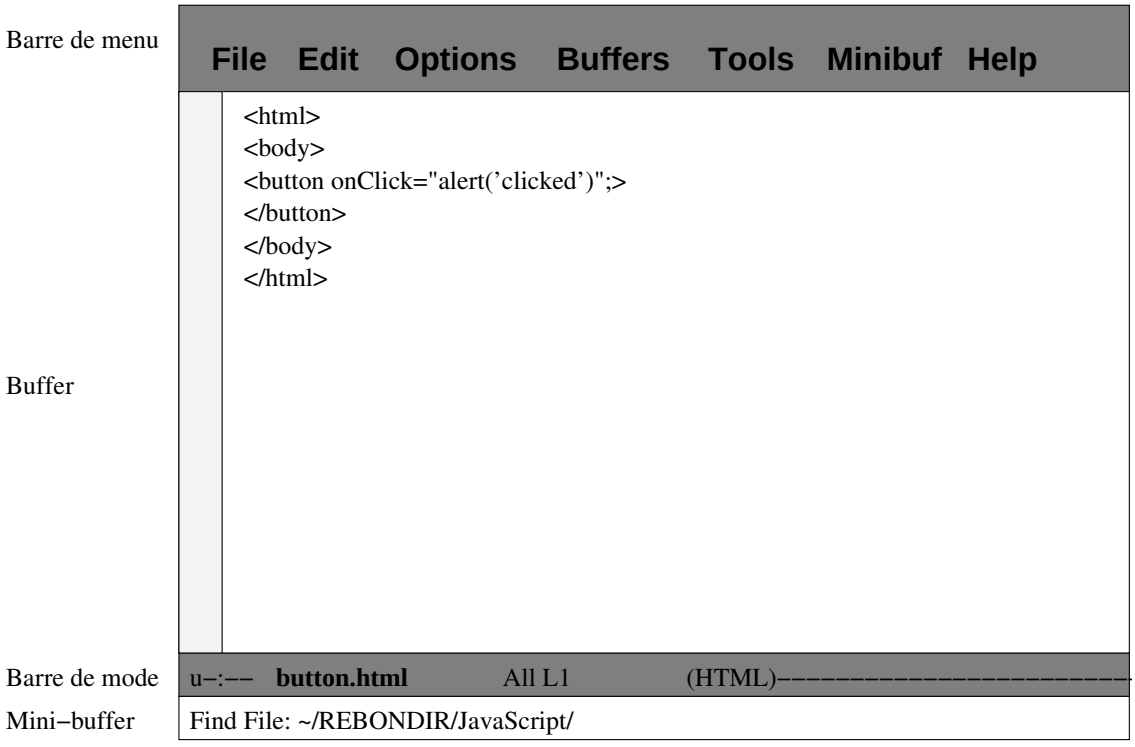
Chapitre 3

Édition de textes avec Emacs

3.1 Lancer et quitter Emacs

Pour lancer Emacs, on peut taper la commande suivante dans une fenêtre Ternimal.

```
$ emacs &
```



Pour sortir de Emacs, taper C-x -C-c (ou utiliser l'onglet Exit Emacs du menu File).

3.2 Commandes, Clés

Chaque *fonctionnalité* (ou *commande*) de Emacs est implémentée par une fonction (EmacsLisp). Les fonctionnalités les plus courantes sont accessibles par des *clés* (touches ou combinaisons de touches) ou à partir des menus. Par exemple,

Commande	Clé	Menu
find-file	C-x C-f	File (Visit New File)
save-buffer	C-x C-s	File (Save)
switch-to-buffer	C-x b	Buffer (item avec le nom du buffer)
mark-whole-buffer	C-x h	Edit (Select All)
save-buffers-kill-emacs	C-x C-c	File (Exit Emacs)

C-x (CONTROL est un préfixe de clé ; quand on tape c-x, Emacs attend la suite.

La clé M-x permet d'appeler n'importe quelle commande à partir de son nom. Par exemple, M-x save-buffer. Le plus souvent, on utilise cette façon de faire quand il n'y a pas de clé associée à la commande. Il y a en effet beaucoup plus de commandes que de clés.

Quand une commande a besoin d'arguments, Emacs les réclame et les lit par l'intermédiaire du Mini-buffer.

3.3 Pour toujours s'en sortir

- La clé C-g associée à la commande abort permet d'abandonner le traitement en cours.
- La clé C-x u associée à la commande undo permet de défaire l'effet de la dernière commande. En répétant, cette commande on peut défaire l'effet de la suite des commandes précédentes.

3.4 Sauvegardes

Quand on charge un fichier f dans un buffer Emacsen vue de le modifier, Emacseffectue une sauvegarde du contenu initial du fichier dans f~.

Au cours de la session, Emacseffectue régulièrement des *auto-sauvegardes* dans le fichier #f#.

3.5 Complétion

Comme bash, Emacs complète grâce à la touche TAB, les noms de commandes, de fichiers. On peut également afficher la liste des complétions possibles avec le caractère ? et choisir une possibilité dans cette liste.

3.6 Aide en ligne

Le préfixe de clé C-h permet d'accèder à l'aide en ligne. En fonction du caractère qu'on tape ensuite on peut obtenir de l'aide sur différents types d'entités.

Entité	Caractère
fonction	f
clé	k
variable	v
mode	m
liaisons clé-commande	b (binding)
...	...

3.7 Paramètre universel

On peut transmettre à une commande un argument en précédant l'appel à la commande par C-u suivi de l'argument

Prenons par exemple la commande `goto-line`. Si nous l'appelons sans argument (`M-x goto-line`), `Emacs` nous demande la valeur de l'argument dans le mini-buffer ; supposons que nous ayons tapé `43` ; le résultat de la commande est le positionnement sur la ligne `43`. Nous pouvons obtenir le même résultat en précédant l'appel à `goto-line` par `C-u 43`.

De façon similaire, on peut utiliser la clé `C-u` pour répéter une commande un certain nombre de fois en commençant par `C-u` suivi du nombre de fois suivi de la commnande à répéter/

Par exemple, `C-u 10 M-x next-line` permet d'avancer de 10 lignes.

3.8 Commandes de base

Voir Emacs Reference Card

3.9 Configuration

Le fichier `~/.emacs` est exécuté à chaque lancement d'`Emacs`. On peut donc configurer `Emacs` en plaçant des commandes `Emacs` dans ce fichier. Le langage utilisé par `Emacs` est l'`EmacsLisp` un dialecte du langage `Lisp`.

Sans connaître le langage `EmacsLisp`, on peut facilement, par mimétisme et en s'inspirant de fichiers `.emacs` existant changer la valeur des paramètres de base.

Le caractère point-virgule (`;`) introduit un commentaire qui se termine enfin de ligne. Les commandes les plus simples sont

1. affecter une valeur à une variable `setq`
2. appeler une commande avec ou sans arguments

```
;; affecter une valeur à une variable
(setq column-number-mode t) ; le numéro de la colonne sera affiché dans la ligne de mode

;; appeler une commande
(global-font-lock-mode t) ; mettre la syntaxe en évidence grâce à des couleurs

(define-key global-map "\M-o" 'goto-line) ; lier la clé M-o à la commande goto-line
```

Si on connaît le langage, on peut rajouter de nouvelles fonctionnalités à `Emacs`.

3.10 Exercices

Exercice 14

Lancer `Emacs`.

- taper `C-x` : `Emacs` attend la suite
- taper `C-f` : `Emacs` commence l'excécution de la commande `find-file` (qui est associée à la clé `C-x C-f`) en affichant dans le mini-buffer le message `Find file: ~/`
- afficher la liste des complétions (`?`)
- taper `/e C-i t C-i` `C-i` est équivalent à TAB)
- en continuant à utiliser la complétion charger le fichier `/etc/passwd`.
- Essayer de modifier ce fichier.
- Détruire le buffer ouvert sur ce fichier.

Exercice 15

Ouvrir le fichier `.bashrc` se trouvant dans votre répertoire d'accueil. Rajouter les `alias` suivants à l'intérieur de ce fichier, à la suite des `alias` déjà existant :

```
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias ll='ls -la'
```

Sauver le fichier.

Exercice 16

Créer un nouveau fichier texte en demandant le chargement d'un fichier qui n'existe pas encore. Taper du texte en explorant les possibilités d'édition. Sauver le fichier. Réouvrir le fichier Regarder le contenu du répertoire.

Exercice 17

Taper du texte dans un nouveau buffer. Sauver le contenu du buffer dans un fichier.

Exercice 18

Construire le damier suivant en utilisant un minimim de clés.

```
XXXX    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX    XXXX
```

Chapitre 4

HTML

4.1 Introduction à HTML

Hyper Text Markup Language : langage à balises hypertexte
Langage de base pour présenter des informations sur Internet.
Toute page Web utilise au moins HTML.
Langage universellement adopté par tous les pays et utilisé dans tous les navigateurs.
Voir le HTML qui produit la page Web courante :

- IE : Affichage/Source
- Netscape : Afficher/ Source de la page ou C-u
- Firefox : Affichage/Code source de la page ou C-u

Une fenêtre s'ouvre et on voit le code HTML source. Une page située à une adresse donnée correspond à un seul et unique code source. Le code HTML des pages Web est souvent produit automatiquement par des logiciels de création de sites Web. Dans ce cas, le code généré est le plus souvent lourd (avec beaucoup de redondances) et difficile à lire pour un humain. En fonction des navigateurs, à l'affichage, le look (couleurs, polices, layout,...) peut changer. Mais le contenu de la page doit être identique quel que soit le navigateur. Le but de ce chapitre est de savoir réaliser une page Web simple en HTML contenant du texte, des images, des tables, des liens vers d'autres pages ou une autre partie de la page courante.

Quand on a modifié le code source d'une page, il faut réactualiser la page pour prendre en compte les changements. Firefox (menu Affichage->actualiser, raccourci-clavier C-r) ou bouton flèche bleue circulaire.

Il y a plusieurs méthodes pour écrire du code HTML. La plus simple est d'utiliser un éditeur de textes. Nous utiliserons celui que nous connaissons : Emacs qui dispose d'un mode adapté à l'édition de code HTML.

4.2 La syntaxe du langage HTML

La syntaxe du langage HTML est basée sur le concept de *balise*. On appelle balise tout mot entouré des signes < et >, comme par exemple <head>, <body>, <img>, <table>, $\cdots$ Il en existe environ une centaine plus ou moins importantes. Elles indiquent la manière d'afficher un texte, une image, un lien, $\cdots$.

Les balises vont en général par paires (balise ouvrante, balise fermante). Entre une balise ouvrante et la balise fermante correspondante, il y peut y avoir (presque) n'importe quel code HTML correct. Si la balise ouvrante est <nom> la balise fermante associée est </nom>. Par exemple, </head>, </body>, </table>, pour presque toutes les balises vues plus haut. Autre exemple : le texte inséré entre les balises et sera affiché en italique (ou en gras avec les balises et).

Certaines balises cependant fonctionnent seules comme la balise `<img />` qui permet d'insérer une image ou `<br />` qui insère un saut de ligne. On trouve aussi la notation `<img>` et `<br>`—pour les balises simples.

Avec ces quelques balises, nous pouvons déjà formater quelque peu un texte.

format.html

```
Bonjour! <br />
Ceci est en <b> caractÃ¨res gras </b> et ceci en <i> italique. </i>

Notez que les lignes blanches n'apparaÃ®ssent pas,
que les sauts de lignes ne se font qu'avec la balise <b> br </b>

et que plusieurs
espaces        sont ramenÃ©s Ã  un seul.
```

4.3 Structure générale d'un document

Un document est généralement structuré de la manière suivante.

```
<html>
<head>
<title> ... </title>
...
</head>
<body>
...
</body>
</html>
```

L'exemple `format.html` précédent était une exception. Dans le cas où les balises `html, head, body` sont omises, le navigateur les rajoute implicitement.

4.4 Corps d'un document

Le corps d'un document `HTML` situé à l'intérieur des balises `body` décrit ce qui s'affiche sur le fond de la page du navigateur.

4.4.1 Insertion d'images

Pour insérer une image dans une page `HTML` il suffit d'utiliser la balise simple `<img />`. Les *attributs* `width, height` servent à redimensionner l'image (dimensions exprimées en pixels). L'attribut `border` permet d'indiquer l'épaisseur de la bordure entourant l'image.

images.html

```
<html>
<head>
<body>
<h4> La bannière </h4>
<img src="http://www.u-bordeaux1.fr/bx1/img-pdf/index/intitule.gif" />
<br>

<h4> Le logo </h4>
<img src="logo-UB1-RVB.jpg" width=100px/>
<br>
<img src="logo-UB1-RVB.jpg" width=50px border=1px />
<br>
<img src="logo-UB1-RVB.jp" width=50px border=1px />
<br>
<img src="logo-UB1-RVB.jp" width=50px border=1px alt"Logo"/>
</body>
</head>
</html>
```

4.4.2 Commentaires

Est un commentaire du langage `HTML` tout texte encadré par les délimiteurs `<!--` et `-->`.

```
<!-- tout ceci est une commentaire -->
```

4.4.3 Ancres (liens)

Les *ancres* ou *liens* permettent de naviguer entre plusieurs pages `HTML` ou au sein d'une même page.

Pour définir un lien on utilise la balise `<a>` (`a` pour *anchor* en anglais). Le résultat du code (texte, image) situé entre la balise de début `<a>` et la balise de fin `</a>` constitue la partie *cliquable* du lien.

La balise `<a>` s'utilise principalement avec l'attribut `href` (pour *hyperlink reference*) qui indique l'URL de la page `HTML` à accéder lors d'un clic sur le lien. La nouvelle page `HTML` peut être

1. une page d'un autre site,
2. une page du même site (dans ce cas par un adressage relatif est possible),
3. la même page mais à un endroit précis.

Par défaut, la nouvelle page s'affiche dans la fenêtre courante. Ce comportement peut être modifié grâce à l'attribut `target`.

Pour pouvoir référencer un endroit particulier d'une page, on positionne un marqueur avec la balise `<a name="etiquette">` où `etiquette` est le nom choisi pour désigner cet endroit. Pour se positionner à cet endroit de la page il faudra faire un lien vers l'adresse relative `#etiquette`.

Toutes ces possibilités sont illustrées dans le fichier `liens.html`.

liens.html

```
<html>
<head>
```

```
</head>
<body>
<a name="debut">
<h4> Lien local </h4>
<a href="format.html"> Lien local </a> <br />

<h4> Lien externe </h4>
<a href="http://www.u-bordeaux1.fr/"> Université Bordeaux1 </a>

<h4> Image cliquable </h4>
<a href="http://www.u-bordeaux1.fr/"> <img src="logo-UB1-RVB.jpg" width=400px border=1px /> </a>

<h4> Ouverture de la page dans une nouvelle fenêtre </h4>
<a href="http://www.u-bordeaux1.fr/" target="_blank"> Université Bordeaux1 </a>
<br>
<center>
<a href="#debut"> Haut de page </a>
</center>
</body>
</html>
```

4.4.4 Formatage du texte

- Balises de titres : `<h1> </h1>`, `<h2> </h2>`, ..., `<h6> </h6>`
- Balises de paragraphe : `<p> </p>`
-

formattage.html

```
<html>
<head>
</head>
<body>
<h1> Ceci est un titre niveau 1 </h1>
<h2> Ceci est un titre niveau 2 </h2>
<h3> Ceci est un titre niveau 3 </h3>
<h4> Ceci est un titre niveau 4 </h4>
<h5> Ceci est un titre niveau 5 </h5>
<h6> Ceci est un titre niveau 6 </h6>
<p>
Un paragraphe commence par une ligne blanche puis se répartit sur plusieurs lignes en
s'adaptant à la taille de la fenêtre avec justification à gauche.
</p>
<p align="center">
Un paragraphe centré s'adapte lui aussi à la taille mais les lignes sont centrées
  sur le milieu de la fenêtre au lieu d'être justifiées à gauche.
</p>
</body>
</html>
```

4.4.5 Tables

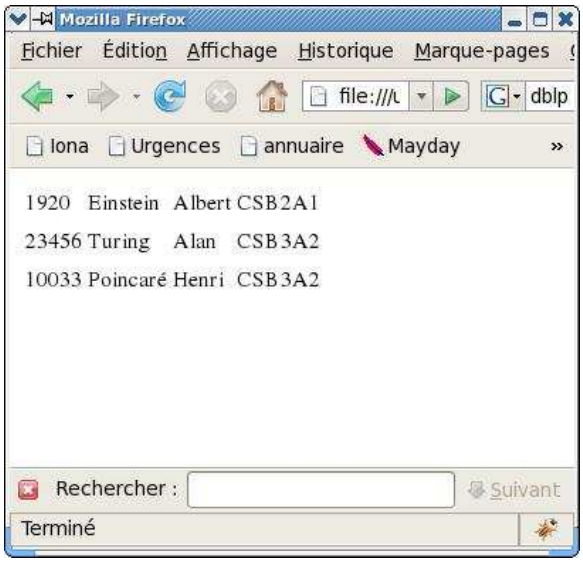


FIG. 4.1 – Page associée au fichier `table1.html`

Les tables sont très utiles pour présenter de manière formatée un ensemble de données ou d'informations.

Par exemple, un ensemble d'étudiants avec leur numéro d'étudiant, nom prénom, nom de groupe pourrait être présenté de la façon comme dans la figure 4.1. Ceci correspond au formattage minimal (organisation en lignes et en colonnes) et est produit par le code du fichier `table1.html`.

Les quatre balises principales sont

`<table>` pour créer une nouvelle table,

`<tr>` (table row) pour créer un nouvelle ligne,

`<td>` (table data) pour créer une donnée dans une ligne,

`<th>` (table header) idem `td` mais pour créer une donnée titre (en gras).

Toutes ces balises ont des attributs qui permettent de changer le "look" de la table : bordures, espacement, alignement,...

`<table>` : border, cellspacing, cellpadding, rules, width, height

`<tr>` :

`<td>` et `<th>` :

4.4.6 Balises diverses

4.4.7 Les formulaires

4.5 Exercices

Exercice 1

Réaliser un document `HTML` avec du texte, un titre, une ancre externe et une ancre interne, une image.

table1.html

```
<html>
<head>
</head>
<body>
<table>
<tr>
<td> 1920 </td> <td> Einstein </td> <td> Albert </td> <td> CSB2A1 </td>
</tr>
<tr>
<td> 23456 </td> <td> Turing </td> <td> Alan </td> <td> CSB3A2 </td>
</tr>
<td> 10033 </td> <td> Poincaré </td> <td> Henri </td> <td> CSB3A2 </td>
</tr>
</table>
</body>
</html>
```

Exercice 2

Réaliser un document contenant une table.

Exercice 3

Réaliser un document présentant un ensemble de photos ou images sous forme de table. Jouer avec les différents attributs pour changer le "look" de la table.

table2.html

```
<html>
<head>
<body>
<table border=1 cellspacing=5 cellpadding=5 width=50% align=center>
<caption> Liste des étudiants </caption>
<tr>
<th> No </th> <th> Nom </th> <th> Prénom </th> <th> Groupe </th>
</tr>
<tr>
<td> 1920 </td> <td> Einstein </td> <td> Albert </td> <td> CSB2A1 </td>
</tr>
<tr>
<td> 23456 </td> <td> Turing </td> <td> Alan </td> <td> CSB3A2 </td>
</tr>
<td> 10033 </td> <td> Poincaré </td> <td> Henri </td> <td> CSB3A2 </td>
</tr>
</table>
</body>
</head>
</html>
```

FIG. 4.2 – Page associée au fichier `table2.html`

Chapitre 5

Les feuilles de style (CSS)

5.1 Introduction

Le langage HTML permet de créer le contenu des pages Web. Les feuilles de style permettent de donner un style à nos pages (couleurs, polices, bordures,...) pour leur donner une apparence plus esthétique.

Il est aussi possible de définir directement des informations de style dans le code HTML mais il est préférable de séparer le contenu du style.

Cela permet par exemple de présenter le même contenu avec différents styles ou d'utiliser le même style pour différentes pages.

Les informations de style seront mises dans un fichier avec l'extension `.css` et chargées dans le fichier HTML en insérant une balise `link` dans la partie *header* du fichier HTML (entre les balises `<head> </head>`.

Exemple :

style.html

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
<h2> Élements h2 en vert et encadrés avec des pointillés </h2>
<p> Paragraphes en bleu et en petits caractères </p>
Le reste avec les valeurs par défaut.
</body>
</html>
```

Si on fait une erreur dans une ligne du fichier de style, elle est simplement ignorée et les valeurs par défaut du navigateur sont utilisées.

5.2 Définition de feuilles de style

Dans une feuille de style, on peut insérer des *commentaires* entre les délimiteurs `/*` et `*/`.

style.css

```
body {
    background-color: yellow;
}

h2 {
   color: green;
   border-style: dotted;
}

p {
  color: blue;
  font-size: 14px;
}
```

On peut (re)définir le style de n'importe quel *type* d'éléments `HTML` en indiquant la valeur d'une ou plusieurs de ses propriétés :

```
élément
{
  propriété1 : valeur1;
  propriété2 : valeur2;
  ...
  propriétéN : valeurN;
```

Par exemple,

```
p {
  font-size: 20px;
  color: blue;
  font-family: arial;
}
```

La propriété est alors modifiée pour tous les éléments de la classe.

Si on veut modifier les propriétés uniquement pour certains des éléments de la classe, on crée une *classe* pour ces éléments.

```
p.bleu {
  font-size: 20px;
  color: blue;
  font-family: arial;
}
```

Dans le texte `HTML` on choisit un élément de cette classe à l'aide de l'attribut `class` :

```
<p class=bleu> ... </p>
```

La classe `bleu` n'est définie que pour les paragraphes (`p`).

5.2.1 Quelques propriétés et leurs valeurs

Propriétés du texte

color	couleur du texte	blue, red, `#F3D5E`, ...
letter-spacing	distance entre les lettres	une distance : 12px, ...
word-spacing		
text-align		left, right, center, justify
text-indent		
text-decoration		
text-transform		
font-size		
font-style		
font-family		
font-weight		
font-variant		

Présentation

background-color	couleur du fond	une couleur
border-style		
border-width		
border-color		
border-top		
border-bottom		

Certaines propriétés sont partagées par plusieurs éléments. On peut définir de nouvelles classes pour préciser ces propriétés.

```
.avec-padding {
  border-style: solid;
  padding-top: 0.5cm;
  padding-left: 0.5cm;
  padding-bottom: 0.5cm;
}
```

On pourra utiliser la classe `avec-padding` avec n'importe quel élément.

```
<p class="encadre"> paragraphe encadre </p>
<img src="logo.gif" class="encadre"/>
<h2 class="encadre"> h2 encadre </h2>
```

5.3 Exercices

Exercice 1

Ajouter un style `CSS` à vos documents déjà réalisés.
Utiliser des classes relatives à un élément particulier ainsi que des classes valables pour plusieurs éléments.

Chapitre 6

JavaScript

6.1 Introduction

Du code `JavaScript` peut être inclus dans des pages `HTML`. Ceci permet entre autre d'apporter du dynamisme aux pages (calculs automatiques, formulaires, ...).

Dans ce cours, nous exécuterons toujours notre code `JavaScript` en l'incluant dans une page `HTML`. Prenons l'exemple de la page `HTML` `fact0.html` qui présente les valeurs de $n!$ (préalablement calculées) pour n allant de 1 à 9.

fact0.html

```
<html>
<head>
<title>
Factorielle
</title>
</head>
<body>
<h2> Table des factorielles </h2>
1! = 1 <br>
2! = 2 <br>
3! = 6 <br>
4! = 24 <br>
5! = 120 <br>
6! = 720 <br>
7! = 5040 <br>
8! = 40320 <br>
9! = 362880 <br>
</body>
</html>
```

Le résultat de ce code est présenté Figure 6.1. Remarquons la duplication de code dans le corps du fichier `HTML`.

Nous allons utiliser `JavaScript` pour calculer et afficher ces résultats automatiquement pour un n quelconque. Pour cela, nous écrivons un bout de code `JavaScript` : tout d'abord, nous initialisons une variable `i` à `0` ainsi qu'une variable `fact` à `1`. Cette dernière est destinée à contenir à chaque étape la valeur de `i!`. Puis nous écrivons une boucle qui va à la fois

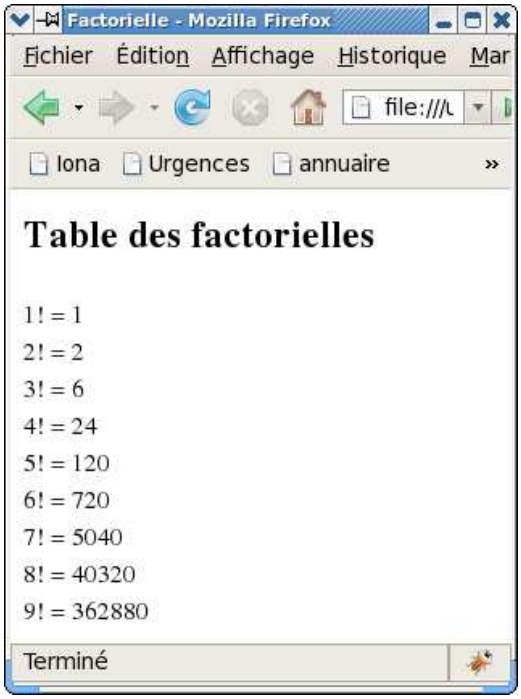


FIG. 6.1 – Page associée au fichier `fact0.html`

1. calculer `i!` à partir de la valeur de `(i-1)!` contenue dans `fact` et sauvegarder cette valeur dans `fact`.
2. afficher le résultat sous la forme souhaitée grâce à l'instruction `document.write`.

fact1.html

```
<html>
<head>
<title>
Factorielle
</title>
</head>
<body>
<h2> Table des factorielles </h2>
<script>
var fact = 1;
var n = 10;
for (i = 1; i < n; i++) {
    fact = fact * i;
    document.write(i + "! = " + fact + "<br>");
}
</script>
</body>
</html>
```

Nous pouvons aussi mettre nos fonctions `JavaScript` dans un fichier séparé, `fact2.js` par exemple.

fact2.js

```
function output_line (i, f) {
    document.write(i + "! = " + f + "<br>");
}

function output_fact (n) {
    var fact = 1;
    var i;
    alert(fact);
    for (i = 1; i < n; i++) {
        fact = fact * ii;
        output_line(i, fact);
    }
}
```

Dans le fichier `HTML` correspondant, le fichier `fact2.js` est chargé lors de l'exécution de la ligne.

```
<script src="fact2.js"></script>
```

Il est alors possible d'appeler la fonction `output_fact` avec comme argument l'entier de notre choix :

```
<script>
output_fact(10);
</script>
```

fact2.html

```
<html>
<head>
<title>Factorielle</title>
<script src="fact2.js"></script>
</head>
<body>
<h2>Table des factorielles</h2>
<script>
output_fact(4);
</script>
</body>
</html>
```

Un des avantages de mettre une fonction dans un fichier séparé est que cette fonction pourra être utilisée par d'autres pages `HTML` ou plusieurs fois dans une même page.

`JavaScript` permet non seulement de contrôler le contenu d'un document `HTML`mais aussi le *comportement* d'un document, c'est à dire la façon dont il réagit aux actions de l'utilisateur (passage de la souris sur un élément, clic de souris, entrée d'une valeur dans un champ de formulaire).

6.2 Environnement de travail

Nos programmes `JavaScript` seront toujours exécutés dans le cadre du chargement d'une page `HTML` dans une fenêtre du navigateur.

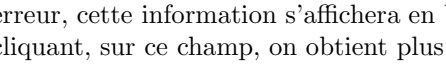
6.2.1 Navigateur Firefox

Comme navigateur, nous utiliserons `Firefox` et pour nous aider à la mise au point de nos programmes, le débogueur `Firebug` (qui est un *plug-in* gratuit de `Firefox`).

En général, pour chaque programme, nous aurons 2 fichiers : le fichier `<nom>.js` qui contient le code `JavaScript` et le fichier `<nom>.html` qui décrit la page `HTML`, charge et utilise le code du fichier `JavaScript`.

6.2.2 Débogueur Firebug

Le langage `JavaScript` n'étant pas interactif (comme `Python` par exemple), il n'est pas aisé de mettre au point un programme sans l'aide d'un débogueur. Nous utiliserons `Firebug` qui est une extension gratuite de `Firefox`.

Après activation de `Firebug`, si le code `JavaScript` chargé par notre page `HTML` contient une erreur, cette information s'affichera en bas à droite de la fenêtre du navigateur 

. En cliquant, sur ce champ, on obtient plus d'information sur l'erreur.

Après correction de l'erreur (ou des erreurs), il faut *recharger* la page `HTML` pour que le programme soit de nouveau exécuté.

À partir de la console de `Firebug` on peut exécuter directement des expressions `JavaScript`.

6.2.3 Récapitulatif

Pour travailler, il nous faudra donc lancer

1. L'éditeur de textes `Emacs` avec un buffer ouvert sur chacun des fichiers `fichier.html` et `fichier.js`.
2. Le navigateur `Firefox` avec une fenêtre ouverte sur `fichier.html`. Pour simplifier, il est préférable d'ouvrir une fenêtre spécifique (au lieu d'utiliser un onglet) pour afficher notre page `HTML`.
3. Le débogueur `Firebug` à partir du menu **Outils** de `Firefox` : **Firebug** → **Ouvrir Firebug**.

6.3 Interaction `JavaScript`-Navigateur

6.3.1 Objet `document`

Un script communique avec le document `HTML` auquel il est associé par l'intermédiaire d'un *objet* nommé `document` qui contient une représentation hiérarchisée du document appelée *Document Object Model (DOM)*. C'est ainsi que l'opération `document.write` affiche son argument sur la

page HTML. À partir de l'objet document, le script a accès à toute la structure et au contenu du document HTML et peut donc le compléter, l'exploiter et le cas échéant le modifier.

JavaScript peut donc faire du *scriptage* de document.

6.3.2 Objet window

JavaScript a accès à la fenêtre du navigateur dans laquelle a été affiché le document HTML par l'intermédiaire de l'objet window qui se trouve au sommet de la hiérarchie de tous les objets JavaScript définis. L'objet window contient entre autres l'objet document qui décrit la structure HTML affichée dans la fenêtre.

Une fois que la page HTML a été affichée (chargée), le navigateur gère les évènements en provenance de la fenêtre (clic sur bouton ou sur lien, envoi dans un champ de formulaire,...). Si des appels JavaScript sont associés à ces évènements, le navigateur demande à JavaScript de les exécuter. Ces appels peut avoir pour effet de modifier le document et de le réafficher, d'ouvrir une nouvelle fenêtre, ...

6.4 Le langage JavaScript

6.4.1 Commentaires

```
// Ceci est un commentaire d'une ligne

/* Celui-ci est aussi sur une ligne */
/* Avec cette syntaxe on peut
   faire des commentaires sur plusieurs lignes */
```

6.4.2 Constantes

Une *constante* est une donnée qui apparaît directement dans un programme.

```
12                    // le nombre 12
1.2                   // le nombre flottant 1.2
"bonjour à tous"      // une chaîne de caractères
'Comment allez vous?' // encore une chaîne de caractères
true                  // la valeur Booléenne "vrai"
false                 // la valeur Booléenne "faux"
```

6.4.3 Variable et affectation

Une *variable* est une case mémoire à laquelle on a donné un nom (un *identificateur*), dans laquelle on peut stocker une donnée et à laquelle on accède grâce à son nom.
Création d'une variable

```
var nom;
```

crée une variable nommée nom, laquelle n'a pour l'instant aucune valeur. On peut aussi déclarer une variable en lui donnant une valeur.

```
var jour = 'lundi';
var année = 2007;
```

Il existe aussi une liste de *mots réservés* (*mots-clés*) qui ne doivent pas être utilisés par le programmeur comme identificateurs car ils ont déjà une signification particulière. En voici la liste :

`break, case, catch, continue, default, delete, do, else, false, finally, for, function, if, in, instanceof, new, null, return, switch, this, throw, true, try, typeof, var, void, while, with`.

Les mots-clés suivants doivent aussi être évités puisqu'ils font partie d'une extension de `JavaScript` en cours de normalisation.

`abstract, boolean, byte, char, class, const, debugger, double, enum, export, extends, final,float, goto, implements, import, int, interface, long, native, package, private, protected, public, short, static, super, synchronized, throws, transient, volatile`.

Un identificateur doit commencer soit par un caractère alphabétique, soit par le caractère `$`, soit par le caractère `_`. Les caractères suivant doivent être alphanumériques, `$` ou `_`.

Exemples : `ma_variable, _temp, $str`.

6.4.4 Expressions arithmétiques

Nombres et Opérateurs arithmétiques : `%, /, *, -, +`

6.4.5 Expressions booléennes

6.4.6 Chaînes de caractères

Chaînes de caractères `""`, concaténation `+`, conversion de type

6.4.7 Opérateurs de comparaison

Égalité

`==`

Inégalité

`!=`

Comparaisons

`<, >, <=, >=`

6.4.8 Opérateurs logiques

Et

`&&`

Ou

`||`

Négation

`!`

6.4.9 Affectation

Affectation d'une valeur à une variable (`=`) :

```
année = 2008;
```

L'ancienne valeur est écrasée. On peut mettre n'importe quelle expression à droite du `=` ; c'est le résultat de l'évaluation de l'expression qui est affecté à la variable ;

```
année = année + 1;
```

Quand un nom de variable apparaît dans une expression il représente sa valeur. Quand un nom de variable apparaît à droite d'un `=`, il représente son adresse en mémoire.

```
var x, y;
x = 10;
y = x + 2;
y = y * 2;
```

6.4.10 Définition de fonction ou procédure

Une *fonction* retourne une valeur. Une *procédure* effectue un traitement (aussi appelé effet de bord) et ne retourne pas de valeur. On entend par traitement, des opérations qui modifient l'environnement extérieur de la procédure (impression à l'écran ou dans un fichier, affectation d'une variable externe à la procédure, ...).

Une fonction retourne une valeur grâce à l'instruction
`return` *expression*

cube.js

```
function cube (n) {
    return n * n * n;
}

function test_cube (n) {
    document.write("<b> cube (" + n + ") </b> = " + cube(n) + " <br>");
}
```

cube.html

```
<html>
<head>
<script src="cube.js"></script>
</head>

<body>
<h3> Début du programme </h3>
<script> document.write("<b> cube (11) </b>= " + cube(11)); </script> <br>
<script> test_cube(12); </script>
<script> test_cube(13); </script>
<h3> Fin du programme </h3>
</body>
</html>
```

6.4.11 Conditionnelles

Instruction `if`

```
function abs (n) {
    if (n >= 0)
        return n;
    else
        return -n;
}
```

Instruction `else if`

Instruction `switch`

6.4.12 Boucles

Boucle `for`

```
function premier (n) {
  if (n % 2 == 0)
    return false;
  for (var d = 3; d * d <= n; d = d + 2)
    if (n % d == 0)
      return false;
  return true;
}
```

Boucle `for/in`

Boucle `while`

```
function decomposition (entier, base) {
    var reste = entier % base;
    var chiffres = [entier % base];

    entier = (entier - reste)/base;
    while (entier > 0) {
        reste = entier % base;
        chiffres = reste + chiffres;
        entier = (entier - reste)/base;
    }
    return chiffres;
}
```

Boucle `do/while`

Instruction `break`

```
for (i = 0; i < tableau.length; i++)
    if (element == tableau[i])
        break;
```

6.4.13 Les Tableaux

Un tableau est une collection ordonnée de variables chacune étant repérée par un indice. Chacune de ces variables est un élément du tableau.

Création d'un tableau

```
var vide = []; // tableau vide
var fibonacci = [1, 1, 2, 3, 5, 8]; // tableau contenanté 6 élments entiers
var notes = ["do", "ér","mi", "fa", "sol", "la", "si"]; // tableau contenant sept îchanes de ècaractres
var divers = [1, "aaa", 1.5]; // tableau avec types édiffrents types de valeurs
```

Accès à un élément

```
var note = t[0];
t[0] = "SOL";
```

6.5 Scriptage de document

Cette technique consiste à utiliser `JavaScript` pour fabriquer un document `HTML`.

Quand dans le corps du document apparaît un appel `JavaScript`, l'appel est effectué et si cet appel effectue des instructions de type `document.write(...)` le document se trouve complété.

C'est exactement ce qui se passe dans l'exemple de présentation des valeurs de factorielle. L'appel `output_fact(10)` produit de nouvelles lignes dans le document `HTML`.

6.6 Les objets

La programmation par objets sort du cadre de ce cours. Cependant, il est impossible de programmer en `JavaScript` sans connaître un tant soit peu la notion d'objet car dans ce langage, en fait, tout est objet. Nous avons déjà évoqué deux objets particuliers `document` et `window` qui font référence respectivement au document associé au programme `JavaScript` et à la fenêtre du navigateur dans laquelle le document s'affiche.

D'autre part, certains objets

6.7 La gestion des évènements

Les évènements proviennent d'objets qui se trouvent sur la page `HTML` (un bouton sur lequel on clique, un champ que l'on renseigne dans un formulaire).

Les traitants des évènements qui peuvent se produire sur un objet `HTML` sont définis par des attributs de cet objet.

6.7.1 Bouton cliquable

button.html

```
<html>
<head>
</head>
<body>
<h3> Début du programme </h3>
<button onclick="alert('You clicked the button')";>
Click Here
</button>
<h3> Fin du programme </h3>
</body>
</html>
```

6.7.2 Entrée d'une valeur

Exemple de programme permettant la saisie d'une valeur en entrée et l'affichage d'une valeur en sortie :

form-fact.html

```
<html>
<head>
<script src="fact.js"> </script>
<script>
function update() {
    document.getElementById("result").innerHTML = fact(document.data.valeur.value);
    document.getElementById("valeur").innerHTML = document.data.valeur.value;
}
</script>
</head>
<body>
<form name = "data">
Value of i: <input type="text" size="5" id="valeur" value = "" onchange="update();"/>
</form>
Fact  i: <span id="result"></span>
</body>
</html>
```

6.8 Exercices

Exercice 1

Écrire une fonction `JavaScript` `test_premier (n)` utilisant la fonction `premier (n)` vue en cours qui affiche si `n` est premier ou pas. Écrire un fichier `HTML` pour tester ces fonctions.

Exercice 2

Écrire un programme `JavaScript` contenant une procédure `afficher_diviseurs (n)` laquelle affiche les diviseurs de l'entier `n`. Écrire un fichier `HTML` qui appelle cette fonction.

Exercice 3

Projet

Écrire un script qui étant donné un répertoire contenant des images fabrique une page qui présente l'ensemble des images sous forme de table.