HTML, JavaScript, PERL

I-HTML

Introduction

Les outils de programmation regroupés dans cette page sont destinés essentiellement aux programmeurs débutants ou amateurs, qui souhaiteraient avoir quelques notions élémentaires en matière de programmation des sites web. Ces outils n'ont donc aucun caractère exhaustif.
Le matériel nécessaire à la réalisation d'une page web est le suivant:
- un ordinateur connecté au réseau internet ;
- un logiciel internet qui peut être Internet explorer ou Mozilla Firefox, Netscape navigator ou Opera, etc ...
- un éditeur de texte qui peut être notepad, ou wordpad, ou microsoftWord, ou wordPerfect, etc.
- windows XP, 2003, linux etc. ...

définitions

www: world wide web

web: un système graphique, interactif, multiplateforme, dynamique (les auteurs peuvent mettre les informations à jour à tout moment, sans que les utilisateurs n'aient besoin de nouveaux équipements) et distribue (c'est a dire que les informations sont reparties et distribuées sur des milliers de sites a travers le monde), qui présente les informations résident sur internet sous forme d'hypertexte, à l'échelle mondiale.

navigateur: logiciel permettant de naviguer, indispensable pour voir, entendre ... le contenu des pages web.

exemple de navigateur: Internet explorer 7.0 ou Mozilla Firefox 1.7, Netscape navigator ou Opera

site web: ensemble de pages situées sur une machine connectée à internet et diffusant un certain type d'informations consultables et reliées de façon cohérentes de façon à donner un type d'informations à l'utilisateur.

1. les bases de HTML

a. définition

HTML: Hyper Text Markup Language.
Le HTML est un langage de description de document (titre, paragraphe, ...) structure. Les différentes parties du document peuvent être étiquetées ou balisées en leur donnant un nom. Ce nom peut-être utilisé dans la suite pour effectuer certaines tâches. Sous HTML, les repères sont appelés *balises* ou *marqueurs* ou *conteneurs*. Tout ce qui n'est pas balise, fait partie du document.
Il existe plusieurs versions de HTML (2.0, 4.0 ...).

b. le fichier HTML et sa structure

Le fichier HTML est un fichier ASCII qui se compose de deux parties:
- le texte du document et
- les balises HTML précisant la structure et les liens hypertextes.

La syntaxe de ces balises est la suivante:

<nom du marqueur>
.
.
.
texte (marqué)
.
.
.
</nom du même marqueur>

Les marqueurs peuvent être écrits en majuscules ou en minuscules
La structure d'un document HTML est la suivante:

<HTML>
.
.

contenu du document

.
.
.
</HTML>

de façon générale, le document a deux parties:

<HTML>

<HEAD>
.

<TITLE> mettre le titre de la page ici </TITLE>
.
</ HEAD >
<BODY>
.
.
.
</BODY>

</HTML>

exercice 1:

exemple de petit programme; saisissez le intégralement sans y ajouter ou retrancher un iota, dans un éditeur de texte: notepad (bloc-notes de microsoft), ou wordpad, ou microsoftWord, ou wordPerfect, etc. ... et **sauvegardez-le au format (c'est à dire avec l'extension) ".html"**. Pour l'exécuter, ouvrez le document à partir de votre navigateur: internet explorer ou netscape navigator, etc. ...

```
<HTML>

    <HEAD>

        <TITLE> exercice 1 </TITLE>

    </HEAD>
    <BODY>

        Mon premier site web

    </BODY>

</HTML>
```

2. la programmation en HTML

a. les commandes de formatage

Il s'agit ici de:

-enrichir le texte (gras ou italique),

-utiliser les textes balisées (espace, tabulation),

-aligner le texte (a gauche, a droite, au centre),

-modifier la taille des polices, etc. ...

style de caractère logique

- pour mettre un texte en italique: *<EM>placer ici le texte à mettre en italique</EM>*
- pour mettre un texte en gras: *<STRONG>placer ici le texte à mettre en gras</STRONG>*
- pour mettre un texte en valeur: *<DFN>placer ici le texte a mettre en valeur</DFN>*

style physique

- gras: *<B>texte à mettre en gras</B>*
- italique: *<I>texte à mettre en italique</I>*
- indice: *_{texte à mettre en indice}*
- exposant: *^{texte à mettre en exposant}*
- retour a la ligne: *
 mettre cette balise pour forcer le retour à ligne*

police de caractères

<FONT SIZE = n FACE = "k">placer ici le texte sur lequel doivent agir les attributs du marqueur FONT</FONT>

NB:

-"n" est un entier qui peut varier de 1 à 7 et représente la taille de la police;

-"k" est une chaîne de caractère qui correspond au nom d'une police de caractère; par exemple, k = Times New Roman, ou k = Verdana, ou k = Fixedsys, ou k = Courier New etc. ...

exercice 2:

```
<HTML>

    <HEAD>

        <TITLE> exercice 2 </TITLE>

    </HEAD>
    <BODY>

        Vous faites vos premiers pas dans la programmation des sites Web <BR>
        Changeons de sujet maintenant; <B> à la une de l'actualité, la question irakienne
        divise qui? </B></I> les grandes puissances</I> <BR>
        <FONT FACE = fixedsys SIZE = 6> le moment est venu d'essayer une autre police de
        caractère, et de varier la taille des caractères </FONT>

    </BODY>

</HTML>
```

b. les liens

Un lien permet de pointer vers un fichier; le fichier s'ouvre lorsque le lien est activé (par exemple en cliquant dessus).
La balise utilisée est la suivante: <A>...</A>
syntaxe: <A>Name = "VarBalise" HREF = "NomFichier"> TITLE = "Titre du lien"</A>

exercice 3:

```
<HTML>

    <HEAD>

        <TITLE> exercice 3 </TITLE>

    </HEAD>
    <BODY>

        je fais déjà des progrès! <A name = "lien1" HREF =
        "C:/DossierWeb/essai1.html">Cliquez ici pour voir s'afficher ma première page
        web</A>
        le bout du tunnel est encore loin!

    </BODY>

</HTML>
```

attention! le nom "C:/DossierWeb/essai1.html" est un exemple; saisissez en lieu et place, le nom que vous avez donne à l'exercice 1
NB: le nom comprend aussi le chemin d'accès au document; ci-dessus, le chemin accès est "C:/DossierWeb/"

c. les tableaux

création:

<TABLE>
.
.
</TABLE>

les attributs de ce marqueur sont:
- *BORDER = n* : épaisseur de la bordure du tableau
- *ALIGN = LEFT/RIGHT/CENTER* : justification du tableau (à gauche, à droite, ou au centre)
- *CELLSPACING = n* : longueur des cellules
- *CELLPADDING = n* : hauteur des cellules
- *WITH = n %*: pourcentage de l'écran que doit occuper le tableau

NB: n est un nombre entier

Il est à noter qu'à ce stade de la création du tableau, vous ne verrez aucun tableau apparaître à l'écran. En effet, lorsque le tableau, qui est en réalité constitué de cellules, est physiquement crée, les cellules n'apparaissent que une à une, au fur et à mesure que vous les remplissez. Pour le remplissage du tableau, d'autres marqueurs sont nécessaires:

- marqueur de ligne: <TR></TR>
- entête des cellules: <TH> placer ici entête des cellules</TH>
- contenu des cellules: <TD> placer ici le contenu des cellules</TD>

Dans chaque cellule, on peut introduire du texte, des chiffres, etc. ...

exercice 4: création du premier tableau

```
<HTML>
    <HEAD>
        <TITLE> Exercice 4 </TITLE>
    </HEAD>
    <BODY>
        <TABLE border = 3 cellspacing = 5 cellpadding = 6 >
            <TR>
                <TD> Dimanche </TD>
                <TD> Lundi </TD>
                <TD> Mardi </TD>
                <TD> Mercredi </TD>
                <TD> Jeudi </TD>
                <TD> Vendredi </TD>
                <TD> Samedi </TD>
            </TR>
```

```
            </TABLE>

        </BODY>

</HTML>
```

exercice 5: création du deuxième tableau

```
<HTML>

        <HEAD>

                <TITLE> Exercice 5 </TITLE>

        </HEAD>

        <BODY>

                <TABLE border=3 cellpadding=14 align=center>

                        <TR>

                                <TH> </TH>
                                <TH> Dimanche </TH>
                                <TH> Lundi </TH>
                                <TH> Mardi </TH>
                                <TH> Mercredi </TH>
                                <TH> Jeudi </TH>
                                <TH> Vendredi </TH>

                        </TR>
                        <TR>

                                <TH> 7h - 7h55 </TH>
                                <TH> algo </TH>
                                <TH> C++ </TH>
                                <TH> Vbasic </TH>
                                <TH> POO </TH>
                                <TH> Réseaux </TH>
                                <TH> Calc num </TH>

                        </TR>
                        <TR>

                                <TH> 8h - 8h55 </TH>
                                <TH> algo </TH>
                                <TH> C++ </TH>
                                <TH> Vbasic </TH>
                                <TH> POO </TH>
                                <TH> Réseaux </TH>
                                <TH> Calc num </TH>

                        </TR>
                        <TR>

                                <TH> 9h - 9h55 </TH>
                                <TH> algo </TH>
                                <TH> C++ </TH>
```

```
                    <TH> Vbasic </TH>
                    <TH> POO </TH>
                    <TH> Réseaux </TH>
                    <TH> Calc num </TH>

                </TR>
                <TR>

                    <TH> 10h - 10h55 </TH>
                    <TH> Roseau </TH>
                    <TH> Calc num </TH>
                    <TH> algo </TH>
                    <TH> maint </TH>
                    <TH> C++ </TH>
                    <TH> maint </TH>

                </TR>
                <TR>

                    <TH> 11h - 11h55 </TH>
                    <TH> réseaux </TH>
                    <TH> Calc num </TH>
                    <TH> algo </TH>
                    <TH> maint </TH>
                    <TH> C++ </TH>
                    <TH> maint </TH>

                </TR>

            </TABLE>

        </BODY>

</HTML>
```

d. images, couleur et fond de page

les images

<IMG SRC = "NomFichier" align = k>

NB: k peut prendre la valeur LEFT ou RIGHT ou CENTER.

couleurs et fond de page

Une couleur est une combinaison des couleurs de base: ROUGE, NOIR, BLANC; en règle générale, l'intensité d'une couleur varie de 0 (intensité qui correspond au NOIR) à 255 (intensité qui correspond au BLANC). Sous HTML, on utilisera les valeurs hexadécimales (qui varient de 00 a FF) de ces intensités. En lieu et place des valeurs hexadécimales, on pourra aussi utiliser les noms anglais des couleurs (BLACK, GREEN, WHITE, etc. ...).

Pour colorier un fond de page, on utilisera l'attribut BGCOLOR **dans** le marqueur *<BODY>* déjà utilise plus haut: *<BODY BGCOLOR = "couleur">*

NB: couleur peut prendre la valeur *BLUE, GREEN, SILVER, GOLD ...*.

II-JavaScript

JavaScript est un langage qui, associé au HTML, permet de rendre les pages dynamiques. Il utilise les variables, les fonctions, les objets, tout comme les langages C++, Java, etc.

1. types variable et donnée

a. les variables

Une variable sert à enregistrer temporairement des données qui peuvent être modifiées lors de l'exécution du programme.
Une variable possède un nom qui permet d'accéder aux données qu'elles contiennent, et un type de données qui définit le genre d'informations pouvant être enregistrées.
En JavaScript, les variables ne doivent pas obligatoirement être déclarées avant leur utilisation; leurs noms suivent les règles suivantes:
- ils doivent commencer par une lettre ;
- leur longueur maximale est de 255 caractères ;
- ils ne doivent pas contenir de point ni de caractère de déclaration de type ;
- ils doivent être unique à l'intérieur d'un domaine de validité

Lorsque les variables sont déclarées, la syntaxe de déclaration est: *var idvar [=valeur]*
NB: *idvar* est le nom de la variable déclarée.

exemple de déclaration d'une variable: numbPers = 0

Il est possible de déclarer plusieurs variables en les séparant d'une virgule.

exemple: x, y, numbPers = 20, prix = 10000, nom = tonguim Les variables sont automatiquement du type de la variable qui leur est affectée.

b. les types de base

- *boolean:* les variables de ce type ne peuvent prendre que les valeurs *true* ou *false*
- *number:* les variables de ce type contiennent des nombres
- *string:* les variables de ce type contiennent des chaînes de caractères.

Il existe des caractères spéciaux pour formater les chaînes de caractères:
- "***\b***": espace arrière
- "***\t***": tabulation
- "***\n***": nouvelle ligne
- "***\r***": retour chariot
- "****": antislash

c. les tableaux

déclaration: var idTab = new array (dim)
NB: *idTab* est le nom de la variable, qui est dans le cas présent un tableau; *dim* est la longueur du tableau.

On peut initialiser le tableau en utilisant la syntaxe: *var idTab = new array ("id1", "id2", ..., "idn").*
L'accès a un champ du tableau, s'effectue en utilisant la syntaxe: *idVar [i]* avec *i = [0; dim-1]*.

2. les opérateurs arithmétiques et les opérateurs comparaison

a. les opérateurs arithmétiques

- "**+**":addition
- "**-**":soustraction
- "***"**:multiplication
- "**/**":division

b. les opérateurs de comparaison

- "**<**": inférieur
- "**>**": supérieur
- "**<=**": inférieur ou égal
- "**>=**": supérieur ou égal
- "**=**": égal
- "**==**": strictement égal
- "***!=***": différent
- "**||**": ou
- "***&&***": et

3. les structures de contrôle

a. les structures conditionnelles

-if (condition)
{instruction 1}
[else
{instruction 2}
]

-for (expression, initialisation; condition sortie; progression)
{instruction}

b. les boucles

- while (condition)
{instruction}

-do
{instruction}
while(condition)

-*break*: force la sortie d'une boucle ou d'un bloc

-switch (expression)
{
case v1: instruction 1; break;
case v2: instruction 2; break;
.
.
.
default::
instructionParDefaut;break;
}

4. la gestion des évènements

La gestion évènements est un mécanisme qui relie une fonction JavaScript, à un évènement tel qu'un click de souris.

Dans la programmation évènementielle, il est important de répertorier tous ces éléments pouvant survenir dans le processus (programme en exécution). Généralement ces évènements se rapportent aux comportements:
- de la souris
- du clavier
- des contrôles (en programmation visuelle, les objets sont tout ce qu'on affiche à écran)
- des fenêtres
- etc. ...

Les évènements les plus utilisés sont:

-par rapport au comportement de la souris on click
on DbClick
on MouseMove
on MouseOut
on MouseOVer
on MouseUp

-par rapport au comportement du clavier on keyPress
on keyDown
on keyUp

-par rapport au comportement des contrôles on Blur
on Form
on Change
on Select

-par rapport au comportement des fenêtres on Resize
on Load
on unLoad

exercice 6: *exemple d'utilisation du HTML et de JavaScript dans un programme*

```
<HTML>

    <HEAD>

        <TITLE> exercice 6 </TITLE>

        <script langage = "JavaScript">

            function afficherAge (nom, prenom, an)
            {

                if(!nom) {nom = "GUINKO"};
                if(!prenom) {prenom = "Ferdinand"};
                if(!an) {an = 1969};
                window.alert (nom + " " + prenom + " vous avez " + (2007 - an) + "
                ans");

            }

        </script>

    </HEAD>
    <BODY>

        <CENTER> <b>Gestion de l'âge par JavaScript</b> </CENTER><br>
        Pour connaître votre age donnez le nom, le prénom et l'année de naissance
        puis cliquez sur le bouton CalculAge ci-dessous <p>
        En cas d'erreur ou pour tout effacer, appuyer sur recommencer <p><p>
        <form name = "varform">

            Nom: <input name = "varNom", size = "30"> <p>
            Prénom: <input name = "varPrenom", size = "28"> <p>
            Année de Naissance: <input name = "varAn", size = "17"> <p>
            <input type = "button" name = "varExe" value = "CalculAge"
            onclick = "afficherAge (document.varform.varNom.value,
            document.varform.varPrenom.value,
            document.varform.varAn.value) "><input type = "reset" name = "varRebegin"
            value = "recommencer">

        </form>

    </BODY>

</HTML>
```

C-après la conception du site

- les serveurs web

a. rôle du serveur web

Un serveur web est un programme situé dans un ordinateur ayant accès à l'internet, et qui attend qu'un navigateur se connecte et lui présente une requête. Le serveur localise alors le fichier demandé et le lui renvoie. On parle alors généralement de serveur et de client web.
Lorsque le programmeur a achevé son site web, il l'installe dans un serveur web, et cela peut se faire de plusieurs façons:

-par l'intermédiaire d'un fournisseur haut débit maîtrisant l'administration d'un serveur
-par l'intermédiaire d'un provider internet, c'est à dire un fournisseur d'accès et de service internet, qui vous indique comment transférer vos fichiers et quel est l'espace disque qui vous alloué ;
-en utilisant des sites appropriés ;
-en créant son propre serveur ;

b. en cas de problèmes

Apres avoir installé son site, des problèmes peuvent surgir; il faut alors:
-vérifier l'écriture du nom du serveur ;
-vérifier les liens; il est préférable d'utiliser des liens relatifs plutôt que des liens absolus ;
-vérifier les autorisations accès ;
-si les images s'affichent mal, vérifier leur extension;

-vérifier la casse (majuscule ou minuscule des liens) et comparer avec les noms réels des fichiers appelés ;

Si les problèmes persistent, appelez l'administrateur système.

c. comment faire connaître votre site web

Pour se faire connaître, il faut se faire référencer par les principaux moteurs de recherche électronique (par exemple, hotBot France, altavista, voila, google, lycos, all the webs) ou par les principaux annuaires électroniques (tel que yahoo, lycos, msn, multimania).

Pour aller plus loin

webographie

le javascript sur le net

1. http://www.editeurjavascript.com
2. http://www.webteacher.com/javascript
3. http://javascript.internet.com
4. http://www.javascriptcity.com
5. http://www.crays.com/jsc/
6. http://www.javascriptsearch.com
7. http://www.javascript-page.com
8. http://www.pageresource.com/jscript

Etc…

le html sur le net

1. http://www.htmlgoodies.com
2. http://www.htmlhelp.com
3. http://www.w3.org/MarkUp/
4. http://www.hwg.org
5. http://www.cwru.edu/help/introHTML/toc.html
6. http://www.mcli.dist.maricopa.edu

Etc…

III-PERL

Introduction

Perl est un langage de programmation créé par Larry Wall en 1986, afin de mettre au point un système de News entre deux réseaux. Il reprend des fonctionnalités du langage C et des langages de scripts sed, awk et shell (sh). PERL est un langage interprété[1] dont l'avantage principal est d'être très adapté à la manipulation de chaînes de caractères. De plus, ses fonctionnalités de manipulation de fichiers, de répertoires et de bases de données en ont fait le langage de prédilection pour l'écriture d'interfaces CGI[2]. Un des atouts de PERL est aussi sa portabilité.
Perl est sensible `a la casse (les majuscules et minuscules sont différenciées), c'est `a dire que **print** est une fonction du langage, alors que **Print** ne l'est pas, mais peut bien sûr être un identificateur (de variables, de fonction, de fichier) `a la charge du programmeur.
Il n'y a pas de formatage particulier dans l'écriture d'une instruction Perl, le séparateur peut être un espace, une tabulation ou même un retour `a la ligne. Les instructions sont séparées entre elles par le caractère **;** (le point virgule) et un commentaire est introduit en plaçant le caractère # `a n'importe quel endroit d'une ligne (mais il ne s'étend que sur une ligne).

Un programme Perl (souvent appelé **script**) débute en général par une ligne qui indique l'endroit où se trouve le compilateur/interpréteur 2, c'est ainsi que l'on peut sur des systèmes Unix démarrer une application Perl en citant simplement son nom. Il n'est pas inopportun de procéder de même sur des systèmes Windows3, du moins dans un environnement WWW/CGI car des serveurs http (tel apache) savent tenir compte de cette première ligne et lancer Perl en fonction de ce qu'elle indique.

1-Les types de données

Une variable est un objet repéré par son nom, pouvant contenir des données, qui pourront être modifiées lors de l'exécution du programme. En PERL, il existe 3 types de variables.
Un nom de variable doit commencer par une lettre (majuscule ou minuscule), précédée d'un "$", un "@" ou un "%"
Un nom de variable peut comporter des lettres, des chiffres et le caractère _ (les espaces ne sont pas autorisés).

Nom de variable correct	Nom de variable incorrect	Raison
$Variable	$nom De Variable	Comporte des espaces
$nom_De_Variable	$123Nom_Variable	Commence par un chiffre
$nom_de_Variable	$toto@mailcity.com	Caractère spécial @
$nom_Variable_123	$Nom-de-variable	Signe interdit

a-Les scalaires ($)

Un scalaire est une donnée atomique. Par exemple, ce pourrait être une chaîne de caractères (suite de caractères) ou un nombre (entier ou flottant). On verra plus tard que les références (c'est-à-dire les pointeurs de Perl) sont des scalaires, même s'ils sont un peu spéciaux. Les variables scalaires sont précédées d'un dollar ($) : $x est donc une variable scalaire.

[1] http://fr.wikipedia.org/wiki/Langage_interpr%C3%A9t%C3%A9
[2] http://www.commentcamarche.net/cgi/cgiintro.php3
http://fr.wikipedia.org/wiki/Common_Gateway_Interface

Instruction	Type de la variable
$Variable = 0;	type entier
$Variable = 12;	type entier
$Variable = 0.0;	type réel
$Variable = 12.0;	type réel
$Variable = 012;	type octal
$Variable = x6A;	type hexadécimal
$Variable = "0.0";	type chaîne
$Variable = "Bonjour tout le monde";	type chaîne

Les chaînes de caractères

PERL a 2 façons de spécifier les chaînes de caractères :

-*par les apostrophes «'»* : dans ce cas, la chaîne n'est pas interprétée
-*par les guillemets «"»* : dans ce cas, la chaîne est interprétée

Exemple :

```
$var2 = 5
print 'Print #1: Il y a $var2 étudiants' ;
print "Print #2: Il y a $var2 étudiants" ;
```

Résultat :

```
Print #1: Il y a $var2 étudiants
Print #2: Il y a 5 étudiants
```

Interprétation des chaînes de caractères

Caractère	Description
\"	guillemets
\\	barre oblique inverse
\n	retour à la ligne
\t	tabulation
\012	caractère codée en octal
\x1A	caractère codée en hexadécimal
\cC	caractère de contrôle (CTRL-C ici)
\$	caractère $

Exemple :

```
$var2=5;
print 'Print #1: Il y a $var2 étudiants\n'."\n";
print "Print #2: Il y a \$var2 étudiants\n";
print "Print #3: Il y a \"$var2\" étudiants\n";
```

Résultat :

```
Print #1: Il y a $var2 étudiants\n
Print #2: Il y a $var2 étudiants
Print #3: Il y a "5" étudiants
```

b-Les tableaux (@)

Les tableaux permettent de stocker plusieurs scalaires en les indiçant. De la même façon qu'en C, on pourra demander le ième élément d'un tableau, i étant un entier. L'accès à un élément sera en temps constant, il ne dépendra pas de son indice dans le tableau. Les variables de type tableau sont précédées d'un arobase (@) : @t est donc une variable de type tableau. Lorsque l'on utilise la syntaxe @t, on désigne la totalité du tableau ; si nous voulons parler du ième élément, nous allons manipuler un scalaire, il faudra donc préfixer par un dollar : $t[4] est l'élément d'indice 4 dans le tableau @t, les crochets [] délimitant l'indice (nul besoin de mettre l'arobase, Perl sait grâce aux crochets qu'il s'agit d'un élément de tableau).

A retenir :

-Les tableaux stockent des données sous forme de liste ordonnée

Exemple :

```
@Tableau = (12,"Bonjour",024);
$Tableau[0] = "Toto $Tableau[1]";
```

-Les tableaux sont accessibles grâce à un index (un numéro représentant l'élément de la liste)
-Il est possible de stocker des éléments de types différents dans un même tableau.
-Une variable tableau commence par le caractère @
-Toutefois lorsque l'on accède à un élément (lorsque le nom du tableau est suivi de crochets), celui-ci est considéré comme une variable et doit donc commencer par le caractère $
-Pour désigner un élément de tableau, il suffit de faire suivre au nom du tableau l'indice de l'élément entre crochets

Exemple :

```
@Tableau = (12,"Bonjour",024);
$Test = $Tableau[1];
$Tableau[0] = "Salut $Tableau[1]";
$test = $Tableau[$#Tableau];
```

-L'occurrence du premier élément est «0»
-Le dernier élément d'un tableau est accessible par la variable $# suivie du nom du tableau (sans espace)

Les fonctions

-Push

```
@elements= ('Orange', 'Pamplemousse', 'Banane');
```

```
$valeur = 'Mangue';
push@elements, $valeur;
# -> @elements= ('Orange','Pamplemousse','Banane','Mangue')
@elements2 = sort @elements;
# -> @elements2 = ('Banane','Mangue','Orange', 'Pamplemousse')
$resultat= join(':',@elements2);
# -> $resultat= 'Banane:Mangue:Orange:Pamplemousse'
@elements3 = split(':',$resultat);
# -> @elements3 = ('Banane','Mangue','Orange','Pamplemousse')
```

-**Localtime**

Application spécifique du tableau. «localtime» retourne une liste de 9 éléments:
@moment = localtime;
-Les secondes
-Les minutes
-L'heure
-Le jour du mois (1 à 31)
-Le mois de l'année (0 à 11, en origine 0)
-L'année (en origine 1900)
-Le jour de la semaine (0 à 6, en origine 0)
-Le jour de l'année (0 à 364), en origine 0)
-L'heure avancée (0 ou 1)

```
($a, $b, $c) = (1, 2, 3);
($a, $b) = (1, 2, 3);
($sec, $min, $heure, $jour, $mois, $an) = localtime;
@moment = localtime;
($sec, $min, $heure, $jour, $mois, $an) = @moment;
print "Moment1 : @moment\n";
print "Moment2 : $heure, $min, $sec\n";
```

```
Moment1 : 58 19 19 9 2 105 3 67 0
Moment2 : 19, 19, 58
```

c-Les tableaux associatifs (hashes) (%)

Une table de hachage en Perl est une structure de données permettant d'associer une chaîne de caractères à un scalaire ; on parle de clefs et de valeurs : une valeur est associée à une clef. Naturellement, dans une même table de hachage les clefs sont uniques ; les valeurs, par contre, peuvent être tout à fait quelconques. Les variables de type table de hachage sont précédées d'un caractère pourcent (%) : %h est donc une variable de type table de hachage. De la même façon que pour les tableaux, %h représente la totalité de la table de hachage ; accéder à un élément se fera avec un dollar : $h{uneclef} est l'élément de clef uneclef de la table de hachage %h, les accolades {} délimitant la clef. Fonctionnellement, on pourrait voir une table de hachage comme un tableau dont les indices peuvent être non-numériques.

Déclaration : %var_asso=("cle1" => "valeur1", "cle2" => "valeur2");

NB : => est équivalent à une virgule
Il n'ya aucune clé en double

Quelques fonctions adaptées aux tableaux associatifs

-keys

La fonction keys(%var hashee) rend une liste des clefs d'un hash.

Exemple:

```
%hash = ('un',1,'deux',2,'trois',3,'quatre',4);
@liste_clef = keys(%hash);
# <==> @liste_clef = ('un','deux','trois','quatre');
```

-values

La fonction values(%var hashee) rend une liste des valeurs d'un hash.

Exemple:

```
%hash = ('un',1,'deux',2,'trois',3,'quatre',4);
@liste_val = values(%hash);
# <==> @liste_val = (1,2,3,4);
```

-each

La fonction each(%var hashee) rend un couple clef,valeur dans une liste `a deux éléments.
Chaque appel de each sur une même variable de type hash rend le couple suivant.

Exemple:

```
%hash = ('un',1,'deux',2,'trois',3,'quatre',4);
($clef,$valeur) = each(%hash);
# au 1er appel <==> $clef='un'; et $valeur=1;
```

each est la fonction adaptée pour parcourir entièrement un hash, elle rend une liste vide lorsque la fin du hash est atteinte.

-undef

Pour réinitialiser une variable :

```
undef$foo;
undef$bar{'blurfl'};
# compare à: delete$bar{'blurfl'};
undef@ary;
undef%hash;
undef*xyz;
# détruit $xyz, @xyz, %xyz, &xyz, etc.
```

-foreach

Cette fonction sert à balayer l'ensemble des variables d'un tableau ou tableau associatif:

Exemple :

```
@elements= (4, 2, 1, 3, 5);
foreach $valeur (@elements) {print "Chiffre: $valeur \n";}
```

Résultat:

Chiffre: 4
Chiffre: 2
Chiffre: 1
Chiffre: 3
Chiffre: 5

foreach (@elements) {print "Chiffre: $_ \n";}

Résultat:

Chiffre: 4
Chiffre: 2
Chiffre: 1
Chiffre: 3
Chiffre: 5

foreach (sort @elements) {print "Chiffre: $_ \n";}

Résultat:

Chiffre: 1
Chiffre: 2
Chiffre: 3
Chiffre: 4
Chiffre: 5

-delete

La fonction delete permet de supprimer un ´el´ement d'un hash en indiquant sa clef.

Exemple:

%hash = ('un',1,'deux',2,'trois',3,'quatre',4);
Delete ($hash {'deux'});

d-Variables d'environnement

Les variables d'environnement permettent d'avoir:
-des informations sur le serveur Web et sa configuration
-des informations sur la requête transmise par le client, via son navigateur Web

Variable	Description
HTTP_HOST	Nom du serveur Web
HTTP_USER_AGENT	Nom du navigateur, et plate-forme du client
PATH_TRANSLATED	Nom du répertoire source du CGI
REMOTE_ADDR	Adresse IP du client
REMOTE_PORT	Numéro de port IP du client
SERVER_ADDR	Adresse IP du serveur Web
SERVER_PORT	Numéro de port IP du serveur Web
REQUEST_METHOD	METHOD : Getou Post
SCRIPT_NAME	Nom du CGI appelé

Toutes les variables d'environnement sont regroupées dans un tableau associatif:
%ENV

foreach$clef(sort keys%ENV)
{

```
        $valeur = $ENV{$clef};
        print "$clef = $valeur \n»;
}
foreach(sort keys%ENV)
{print "$clef = $_ \n»;}
```

2-Les opérateurs

a-Les opérateurs de calcul

Ils permettent de modifier mathématiquement la valeur d'une variable

Opér.	Dénomination	Effet	Ex.	Résultat (avec $x valant 7)
+	opérateur d'addition	Ajoute deux valeurs	$x+3	10
-	opérateur de soustraction	Soustrait deux valeurs	$x-3	4
*	opérateur de multiplication	Multiplie deux valeurs	$x*3	21
/	opérateur de division	Divise deux valeurs	$x/3	2.3333333
=	opérateur d'affectation	Affecte une valeur à une variable	$x=3	Met la valeur 3 dans la variable $x

b-Les opérateurs d'assignation

Ils permettent de simplifier des opérations

Exemple:

$x=$x+2 peut s'écrire $x+=2

Opér.	Effet
+=	addition deux valeurs et stocke le résultat dans la variable (à gauche)
-=	soustrait deux valeurs et stocke le résultat dans la variable
*=	multiplie deux valeurs et stocke le résultat dans la variable
/=	divise deux valeurs et stocke le résultat dans la variable

Exemple :

-Affectation et opération combinée

```
$var += 2;          <==> $var = $var + 2
$var *= 3;          <==> $var = $var * 3
$chaine .= 'xxx'; <==> $chaine = $chaine . 'xxx';
```

c-Les opérateurs d'incrémentation

Opér	Dénomination	Effet	Syntaxe	Résultat : avec x valant 7
++	Incrémentation	Augmente d'une unité la variable	$x++	8
--	Décrémentation	Diminue d'une unité la variable	$x--	6

Exemple :

$I = 10;
$I++; <==> $I = $I + 1; donc I = 11
$K = ++$I; <==> $K = $I = $I + 1; donc K = I = 12
$K= $I++; <==> $K = $I; $I = $I + 1;
donc K = 12 et I = 13

d-Les opérateurs de comparaison

Opérateur	Dénomination	Ex.	Résultat (avec x valant 7)
==	opérateur d'égalité	$x==3	Retourne 1 si $x est égal à 3, sinon 0
<	opérateur d'infériorité stricte	$x<3	Retourne 1 si $x est inférieur à 3, sinon 0
<=	opérateur d'infériorité	$x<=3	Retourne 1 si $x est inférieur à 3, sinon 0
>	opérateur de supériorité stricte	$x>3	Retourne 1 si $x est supérieur à 3, sinon 0
>=	opérateur de supériorité	$x>=3	Retourne 1 si $x est supérieur ou égal à 3, sinon 0
!=	opérateur de différence	$x!=3	Retourne 1 si $x est différent de 3, sinon 0

e-Les opérateurs de comparaison de chaînes

Opér	Dénomination	Exemple	Résultat (avec x valant 7)
.	opérateur de concaténation	$x." chaîne"	Ajoute la chaîne " chaîne" à $x
eq	opérateur d'égalité	$x eq "Bonjour"	Retourne 1 si $x contient la chaîne "Bonjour", sinon 0
lt	opérateur d'infériorité stricte	$x lt "Toto"	Retourne 1 si $x est inférieure à "Toto"; 3, sinon 0
le	opérateur d'infériorité	$x le "Toto"	Retourne 1 si $x est inférieure ou égale à "Toto", sinon 0
gt	opérateur de supériorité stricte	$x gt "Toto"	Retourne 1 si $x est supérieure à "Toto", sinon 0
ge	opérateur de supériorité	$x ge "Toto"	Retourne 1 si $x est supérieure ou égale à "Toto", sinon 0
ne	opérateur de différence	$x ne "Toto"	Retourne 1 si $x est différente de "Toto", sinon 0

f-Les opérateurs logiques (booléens)

Opér	Dénomination	Effet	Syntaxe
||	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

3-Les structures de contrôles

Diverses structures de contrôle sont proposées en Perl, elles évaluent une expression et proposent un traitement selon les schémas algorithmiques classiques. L'expression à évaluer est convertie en chaîne, une chaîne non vide ou différente de "0" correspond à la valeur **vrai**. Les opérateurs de comparaison (==,>,gt,...) retournent 1 pour vrai et 0 pour faux ("0" lorsque converti en chaîne), ci-dessous quelques exemples d'expressions vraies ou fausses :

0 <==> faux
"0" <==> faux, c'est 0 converti en chaîne
'' <==> faux, chaîne vide
'00' <==> vrai, différent de 0 ou '0'
1 <==> vrai
"n'importe quoi" <==> vrai
Undef <==> undef rend une chaîne vide donc faux

Perl permet de structurer des instructions en utilisant des blocs délimités comme en C par les accolades (**{ }**). La structuration en blocs a un impact sur la portée des variables

a-L'instruction if

La syntaxe générale de l'instruction **if** est la suivante :

```
if (expression_à_évaluer)
{
        instruction(s) à exécuter si l'expression est vraie
}
else
{
        instructions à exécuter si l'expression est fausse
}
```

Si il n'y a rien de particulier à faire dans le cas ou l'expression est fausse, on peut se passer du **else** {**...**}. Par contre même s'il n'y a qu'une seule instruction à exécuter, il convient de la placer dans un bloc {}.

```
if ($I == 0) { $I++; } <==> le else est omis
if ((10 == 10) == 1) { ... } <==> vrai car == rend 1 si la
condition est réalisée
```

Le test de plusieurs cas peut être enchaîné en utilisant **elsif**:

```
if ($Couleur eq "Bleu") { ... }
elsif ($Couleur eq "Rouge") { ... }
elsif ($Couleur eq "Vert") { ... }
else { ... }
if ($Couleur eq "Noir") { ... }
else if ($Couleur eq "Jaune") <==> erreur de syntaxe, un else doit être suivi par une accolade {
```

b-L'instruction UNLESS

L'instruction **unless** fonctionne comme un **nif** (non if : si l'expression est fausse alors faire).

```
unless ($I > 0) { ... }
else { ... }
```

Un **elsif** peut suivre un **unless**, le **elseunless** n'existe pas.

c-Les instructions while et until

Perl dispose d'une structure *tant que* (**while**) et d'une structure *jusqu'à ce que* (**until**).

```
$I = 0;
while ($I < 10) { $I++; } <==> tant que I est inférieur à 10

$I = 0
until ($I == 10) { $I++; } <==> jusqu'à ce que I soit égal à 10
while (1) { ... } <==> sans fin
while () { ... } <==> sans fin, une expression inexistante n'est pas vide !
until (0) { ... } <==> sans fin
```

Pour forcer la sortie d'une boucle **while** ou **until** sans s'occuper de la condition d'arrêt, on utilise l'instruction **last** (de la même fac¸on qu'un **break** en Langage C). Pour faire évaluer la condition sans avoir déroulé la totalité des instructions du bloc, on utilise l'instruction **next**.

d-L'instruction for

L'instruction **for** correspond à une boucle *pour* dans laquelle on fournit une valeur de départ, une expression à évaluer et une expression permettant une réinitialisation (incrément, décrément, ...).

```
for ($I=0; $I < 10; $I++) { ... }
```

Dans l'exemple ci-dessus, on affecte 0 à I, on évalue ($I < 10), si c'est vrai on exécute les instructions incluses dans le bloc et on passe à l'incrémentation de I, si c'est faux on passe à l'instruction suivante.

for (;;) { ... } <==> boucle sans fin, les expressions ne sont pas obligatoires

Les instructions **next** et **last** fonctionnent de la même manière que pour **while** et **until.**

e-L'instruction foreach

L'instruction **foreach** charge dans une variable les valeurs successives d'une liste et effectue le traitement sp´ecifi´e dans le bloc d'instructions. Quand la fin de la liste est atteinte, on passe à l'instruction suivante :

```
foreach $Couleur ('jaune','vert','bleu') { ... }
la variable couleur prend successivement les valeurs jaune, vert et bleu
```

foreach $var (@tableau1, @tableau2, $I, $K, 10)
la variable $var prend successivement toutes les valeurs scalaires contenues dans $tableau1 ... jusqu'à la valeur 10

Ici aussi, **last** provoque une sortie forcée de la boucle et **next** un passage à la valeur suivante.

f-Exécuter si l'expression précédente est vraie ou fausse

L'opérateur **&&** permet d'évaluer une expression si celle qui la précède est vraie, l'opérateur || permet d'évaluer une expression si celle qui la précède est fausse :

```
$J = $I % 2 && printf ("I est impair \n");
I modulo 2 vaut 0 ou 1
si 1 le nombre est impair
si 0 le nombre est pair
$J = $I % 2 || printf ("I est pair \n");
```

Les opérateurs **&&** et || peuvent être utilisés pour former des expressions conditionnelles évaluées par l'instruction **if** :

```
if (($a == $b) || ($a == $c)) { ... }
```

Une dernière construction est possible, elle admet 3 expressions séparées par les opérateurs**?** et **:**, la seconde est évaluée si la première est vraie, la troisième est évaluée si la première est fausse :

```
$J = $I % 2 ? printf ("impair \n") : printf ("Pair \n");
```

A retenir :

Boucle	Description
foreach	Répétition d'un bloc pour chaque élément de la liste
while	Répétition d'un bloc tant que la condition est vraie
until	Répétition d'un bloc tant que la condition n'est pas vraie
for	Répétition d'un bloc un certain nombre de fois
do	Exécution d'un bloc au moins une fois

4-Entrée standard et sortie standard

Compte tenu du passé Unix de Perl, on lit sur le STDIN et on écrit sur le STDOUT. Par défaut il s'agit du clavier et de l'écran, mais Perl ne s'en occupe pas, c'est au système d'exploitation de lui fournir des descripteurs valides.

a-Lecture sur l'entrée standard

L'opérateur de lecture sur l'entrée standard est <**STDIN**>.

$ligne = <STDIN>; <==> on affecte à la variable $ligne

la ligne suivante

@lignes = <STDIN>; <==> le tableau @lignes reçoit un élément par lignes lues

Pour lire et traiter ligne à ligne on procède en général comme suit :

while ($ligne = <STDIN>) Lorsque la dernière ligne est { ... } atteinte,
<STDIN> retourne undef
et la boucle while s'arrête

Perl autorise une autre forme de lecture du STDIN avec l'opérateur <> qui passe au programme tous les fichiers indiqués sur la ligne de commande (chaque argument de la ligne de commande est alors considéré comme un nom de fichier).

b-Ecriture sur la sortie standard

L'instruction **print** écrit une liste de chaîne de caractères sur la sortie standard.

print (@liste1, $chaine, @liste2, 'toto');

Il est possible de contrôler la présentation des impressions sur la sortie standard en utilisant l'instruction **printf** qui fonctionne comme en **C**. Elle admet comme argument une liste dont le premier élément est une chaîne de caractères qui contient le masque d'impression des variables ainsi que des valeurs constantes à intercaler entre elles (les autres éléments de la liste sont les variables à imprimer).

printf ("%s %s %5d \n",$chaine1,$chaine2,$I);
%s est le masque d'impression de $chaine1
%s est le masque d'impression de $chaine2
%5d est le masque d'impression de $I
\n constante qui introduit une nouvelle ligne

Si un argument de **printf** est une variable tableau, il convient de spécifier autant de masques d'impression que de valeurs scalaires distinctes contenues dans le tableau. Les masques d'impression indiquent les conversions à réaliser et s'utilisent comme dans l'exemple suivant :

%10s <==> une chaîne sur sa longueur réelle complétée éventuellement par des espaces (10 maxi)
%.5s <==> les 5 premiers caractères d'une chaîne
%3d <==> un entier tel que ou sur 3 caractères s'il est inférieur à 99 (espace à gauche)
%o <==> un nombre en octal
%x <==> un nombre en hexadécimal
%f <==> un réel avec 6 décimales
%.3f <==> un réel avec 3 décimales
%e <==> un réel en représentation mantisse/exposant

c-Printf

En plus de print, on peut utiliser printf. Elle est similaire à printf en langage C. Voici la syntaxe avec quelques paramètres disponible avec cette fonction:

printf format, liste;
format : %m.nx

m : largeur (optionnel)
n : précision (optionnel)
x : s chaîne de caractères
c caractère
d nombre décimal
x nombre hexadécimal
o nombre octal
f nombre en point flottant en format fixe

Utilisation de Printf	Résultat
$a = 20; $b = 3; $c = $a / $b; print "\$c =$c\n"; printf "\$c =%3.2f\n",$c;	 $c =6.66666666666667 $c = 6.67

d-Lire un fichier

On peut aussi lire un fichier comme suit:

```
open(FILE,"/etc/passwd");
while (<FILE>)
{
        chomp($_);
        print "> $_ <\n";
}
close(FILE);
```

Si on veut faire une analogie avec une lecture au terminal, <FILE> est similaire à <STDIN>. Plus haut, nous avons vu que pour lire une donnée au terminal, on utilise <STDIN>. Donc <FILE> permet de lire une ligne à la fois dans un fichier. Pour déterminer la fin d'une ligne, comme en langage C, on utilise les caractères \0. Donc le programme ci-dessus donnera:

```
> root:x:0:0:root:/root:/bin/bash <
> bin:x:1:1:bin:/bin: <
> daemon:x:2:2:daemon:/sbin: <
> adm:x:3:4:adm:/var/adm: <
> lp:x:4:7:lp:/var/spool/lpd: <
> sync:x:5:0:sync:/sbin:/bin/sync <
> shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown <
> halt:x:7:0:halt:/sbin:/sbin/halt <
> mail:x:8:12:mail:/var/spool/mail: <
> news:x:9:13:news:/var/spool/news: <
> uucp:x:10:14:uucp:/var/spool/uucp: <
> operator:x:11:0:operator:/root: <
> games:x:12:100:games:/usr/games: <
> gopher:x:13:30:gopher:/usr/lib/gopher-data: <
> ftp:x:14:50:FTP User:/home/ftp: <
> nobody:x:99:99:Nobody:/: <
> xfs:x:100:101:X Font Server:/etc/X11/fs:/bin/false <
> gdm:x:42:42::/home/gdm:/bin/bash <
```

e-Écrire dans un fichier

Exemple d'écriture dans un fichier :

```
print "Entrez votre prenom : ";
        $prenom = <STDIN>;
        chomp($prenom);
print "Entrez votre nom : ";
        $nom = <STDIN>;
        chomp($nom);
open(FILE,"> /tmp/fichier.out");
print FILE "Hello $prenom $nom\n";
close(FILE);
```

Dans l'énoncé « open » ci-dessus, on utilise le "FILEHANDLE" accompagné du caractère « > » afin d'écrire dans le fichier. Le programme ci-dessus saisit un prénom et un nom, et inscrit le résultat dans le fichier /tmp/fichier.out.

Résultat :

Entrez votre prenom : Ferdinand
Entrez votre nom : GUINKO
Le fichier /tmp/fichier.out contient alors ceci:
Hello Ferdinand GUINKO

4-La variable $

Perl utilise une variable scalaire fourre-tout nommée **$** ($ souligné) comme résultat par défaut d'un certain nombre d'instructions. C'est notamment le cas avec la lecture sur l'entrée standard :

while (<STDIN>) <==>
{ <==>
chop; <==>
} <==>
	while ($_ = <STDIN>)
	{
	chop($_);
	}

La variable **$** est également implicitement utilisée par les instructions **foreach** et **print**[3], ainsi que par les instructions qui manipulent des expressions régulières[4] ou celles qui testent l'existence de fichiers et répertoires

```
@liste=(1,2,3,4,5);
```

foreach (@liste) <==>
{ <==>
print ; <==>
} <==>
	foreach $_ (@liste)
	{
	print $_;
	}

Exercices:

Exercice 1

```
#!c:/perl/bin/perl.exe
# Exercice 1
# Lecture de nombres, pour chacun d'entre eux on indique
# s'il est pair ou impair. `A l'issue de la saisie (Ctrl-d
# sous Unix ou Ctrl-z sous Windows) on affiche la moyenne
# des nombres entr´es.
print ("Entrer un nombre \n");
$n=0;
$total=0;
while(<STDIN>)
{
```

[3] Mais printf n'utilise pas la variable **$** par défaut

[4] http://fr.wikipedia.org/wiki/Expressions_rationnelles
http://www.commentcamarche.net/php/phpreg.php3
http://www.expreg.com/

```
	# le résultat de la saisie dans $_
	chomp(); # enlève le délimiteur de fin de ligne
	if ($_ % 2)
	{ # rend 1 si impair et 0 sinon
		print ("$_ est impair \n");
	}
	else
	{
		print ("$_ est pair \n");
	}
	$n++; $total += $_;
	print ("Entrer un nombre \n");
	}
	if ($n) {
	print ("La moyenne est : ",$total / $n,"\n");
}
```

Exercice 2

```
#!c:/perl/bin/perl.exe
# Exercice 2
#
# Constituer une liste de mots entrés au clavier l'un après l'autre.
# Trier la liste constituée par ordre croissant puis par ordre décroissant.
#
print ("Entrer un mot \n");
while (<STDIN>)
{
	chomp();
	push (@liste, $_);
	print ("Entrer un mot \n");
}

if (@liste)
{
	@tri=sort(@liste);
	@rev=reverse(@tri);
	print("Par ordre croissant : \n");
	print ("@tri \n");
	print ("Par ordre decroissant : \n");
	print ("@rev \n");
}
```

Exercice 3

```
#!c:/perl/bin/perl.exe
# Exercice 3
#
# On utilise un Hash pour gérer un stock. Le nom d'un
# vin est utilisé comme clef, une valeur saisie au clavier
# vient en + ou - de l'éventuelle valeur déjà existante.
#
%Stock=();
print ("Nom du vin : \n");
while ($Vin=<STDIN>)
{
	#Vin sera la clef du Hash de stock.
	chomp($Vin); # Enlève le car de fin de ligne.
	print ("Quantit´e : ");
	$Quant=<STDIN>; # Il faudrait vérifier qu'il s'agit
```

```
        # bien d'un nombre entier relatif.
        chomp($Quant);
        unless ($Quant)
        {
                # Si l'utilisateur ne saisit pas de
                last; # quantité, on affiche le stock.
        }
        $Stock{$Vin} += $Quant;
        print ("Nom du vin : \n");
}
# Parcourir le stock
while (($Vin,$Quant) = each(%Stock))
{
        print ($Vin,"=",$Quant,"\n");
}
```

5-Expressions régulières

Voici quelques exemples d'expression régulière:

```
/a[bc]d/          # correspond aux chaînes abd et acd
/[a-z]/           # correspond à tous les caractères entre a et z
/a[^bc]d/         # correspond à a.d sauf abd et acd
/arti(chaut|ste)/ # correspond à artichaut et artiste
```

Exemples de quantificateur:

```
$var = "abracadabra";
$var =~ /a(.+)a/;
print "\$1 = $1\n"; # $1 contient bracadabr
$no_tel =~ /^[0-9]{3}-[0-9]{4}$/; # numéro de téléphone
```

Exemples de substitution:

```
$var = "artichaut";
$var =~ s/chaut/ste/; # artichaut devient artiste
$var = "artichaut artiste";
$var =~ s/(chaut|ste)/cle/g; # "article article";
```

Exemples sur l'opérateur de translation:

```
$var = "artichaut";
$var =~ tr/[a-z]/[A-Z]/; # artichaut devient ARTICHAUT
```

a-Exemples d'expression régulière

Utilisation des méta-caractères:

. "point", signifie n'importe quel caractère, excepté newline
[ ] classe de caractères: choisir 1 et 1 seul des caractères à l'intérieur des [ ]
/a./ La chaîne de caractères **doit contenir** le caractère "a" suivi de n'importe quel caractère

/a.b.c/ La chaîne de caractères **doit contenir** le caractère "a" suivi de n'importe quel caractère suivi du caractère "b" suivi de n'importe quel caractère suivi du caractère "c"

Exemple: axbxc ou a&b&c

/[aeiouyAEIOUY]/ La chaîne de caractères **doit contenir** n'importe laquelle de ces voyelles

/[0123456789]/ La chaîne de caractères **doit contenir** n'importe laquelle des chiffres entre 0 et 9

/[0-9]/ La chaîne de caractères **doit contenir** n'importe laquelle des chiffres entre 0 et 9

/[a-z]/ La chaîne de caractères **doit contenir** n'importe laquelle des lettres minuscules de a à z

Utilisation des méta-caractères:

| alternative (à placer entre parenthèses)

/([a-z] | [A-Z])/ La chaîne de caractères **doit contenir** une minuscule "a à z" ou une majuscule "A à Z"

/(auto | maison)/ La chaîne de caractères **doit contenir** le mot "auto" ou le mot "maison"

/(0[1-9] | 1[0-2])/ La chaîne de caractères **doit contenir**: 01 à 09 ou 10 à 12 Autrement dit: les chiffres 01 à 12

Utilisation des méta-caractères:

^ Début de la chaîne de caractères
$ Fin de la chaîne de caractères

/Bonjour/ La chaîne de caractères **doit contenir** le mot Bonjour

/^Bonjour$/ La chaîne de caractères **doit contenir seulement** le mot Bonjour

/^Bonjour/ La chaîne de caractères **doit commencer** par le mot Bonjour

/Bonjour$/ La chaîne de caractères **doit se terminer** par le mot Bonjour

/^(0[1-9]|1[0-2])$/ La chaîne de caractères **doit contenir seulement**: 01 à 09 ou 10 à 12, autrement dit: les chiffres 01 à 12

/(0[1-9]|1[0-2])/ La chaîne de caractères **doit contenir**: 01 à 09 ou 10 à 12, autrement dit: les chiffres 01 à 12

/^(0[1-9]|1[0-2])/ La chaîne de caractères **doit commencer seulement** par: 01 à 09 ou 10 à 12, autrement dit: les chiffres 01 à 12

/(0[1-9]|1[0-2])$/ La chaîne de caractères **doit se terminer seulement** par: 01 à 09 ou 10 à 12, autrement dit: les chiffres 01 à 12

/^(auto|maison)/ La chaîne de caractères **doit commencer** par le mot "auto" ou le mot "maison"

/^([a-z]*|[A-Z]*)$/ La chaîne de caractères **doit contenir seulement** des lettres minuscules ou des lettres majuscules.

Utilisation des quantificateurs:

***** Apparaît 0, 1 ou plusieurs fois
+ Apparaît 1 ou plusieurs fois
? Apparaît 0 ou 1 fois

/^Bananes?$/ La chaîne de caractères **doit contenir** seulement le mot Banane (singulier ou pluriel), la lettre "s", 0 ou 1 fois

/^[01]?[0-9]$/ La chaîne de caractères **doit contenir** seulement: 0 à 9 ou 00 à 09 ou 10 à 19, autrement dit: les chiffres 0 à 19, avec 1 ou 2 chiffres

/^a.*a$/ La chaîne de caractères **doit commencer** par la lettre "a" et se terminer par la lettre "a"

/^a.*toto.*a$/ La chaîne de caractères **doit commencer** par la lettre "a", avoir n'importe quel caractère (0, 1 ou plusieurs), avoir le "toto", avoir n'importe quel caractère (0, 1 ou plusieurs) et se terminer par la lettre "a"

/^a.+a$/ La chaîne de caractères **doit commencer** par la lettre "a", avoir n'importe quel caractère (1 ou plusieurs) et se terminer par la lettre "a"

/^a.+toto.+a$/ La chaîne de caractères **doit commencer** par la lettre "a", avoir au moins un caractère quelconque, avoir le "toto",

Exercice 4

```
# #!c:/perl/bin/perl.exe
# Exercice 4
#
# Calcul d'expressions arithmétiques simples (non parenthésées) saisies à la volée.
#
# La chaîne doit commencer par un nombre
# suivi d'un opérateur arithmétique suivi d'un nombre (n fois).
# On admet que eval(expr) fait le calcul.
print ("Entrer une expression `a calculer \n");
while (<STDIN>)
{
        if (/^\d+([+*\/-]\d+)+$/)
        {
                # si le - situé entre [] n'était pas positionné
                # juste avant ], il faudrait le précéder d'un
                # / car il serait alors pris comme délimiteur de
                # classe (cf. [a-z]).
                print ($_," = ",eval($_),"\n");
        }
        else
        {
                print ("Expression arithmétique incorrecte \n");
        }
        print ("Entrer une expression `a calculer \n");
}
```

Exercice 6

```
#!c:/perl/bin/perl.exe
# Exercice 6
#
# On vérifie que des nombres entrés au clavier sont
# bien des entiers relatifs ou des réels.
#
```

```
print ("Entrer un nombre : \n");
while (<STDIN>)
{
	chomp();
	if (/^[+|-]?\d+$/)
	{
		# commence éventuellement par un + ou un -
		# suivi et terminé par au moins un chiffre.
		print ("$_ est un nombre entier relatif \n");
	}
	elsif (/^[+-]?\d+\.\d+$/)
	{
		# commence éventuellement par un + ou un -
		# suivi par au moins un chiffre suivi par
		# un . suivi et terminé par au moins un
		# chiffre.
		print ("$_ est un nombre r´eel \n");
	}
	else
	{
		print ("$_ n'est pas un nombre \n");
	}
	print ("Entrer un nombre : \n");
}
```

Exercice 5

```
# #!c:/perl/bin/perl.exe
# Exercice 5
#
# On entre un texte à la volée.
# Lorsque la saisie est terminée, on calcule le nombre de lignes,
# de mots et de caractères du texte.
#
printf ("Entrer un texte libre sur plusieurs lignes \n");
@texte=<STDIN>;
chomp(@texte);
$nbmots = 0;
$nblig = 0;
$nbcar = 0;
foreach (@texte)
{
	$nblig++;
	@mots=split();
	$nbmots = $nbmots + @mots;
	foreach (@mots)
	{
		$nbcar += length(); # length($-)
	}
}
print ("Nb de lignes saisies : $nblig \n");
print ("Nb de mots saisis : $nbmots \n");
print ("Nb de caractères saisis : $nbcar \n");
```

6-Liens utiles

-http://www.perl.com
-http://www.activestate.com
-http://www.perl.org (tout sur Perl)
-http://www.perl.com (tout sur Perl également)
-http://www.cpan.org (distribution de Perl et des modules)
-ftp://ftp2.developpez.be/developps/perl/perl-all-fr-pdf.pdf
-http://lhullier.developpez.com/tutoriels/perl/intro/
-http://perl.enstimac.fr/DocFr.html