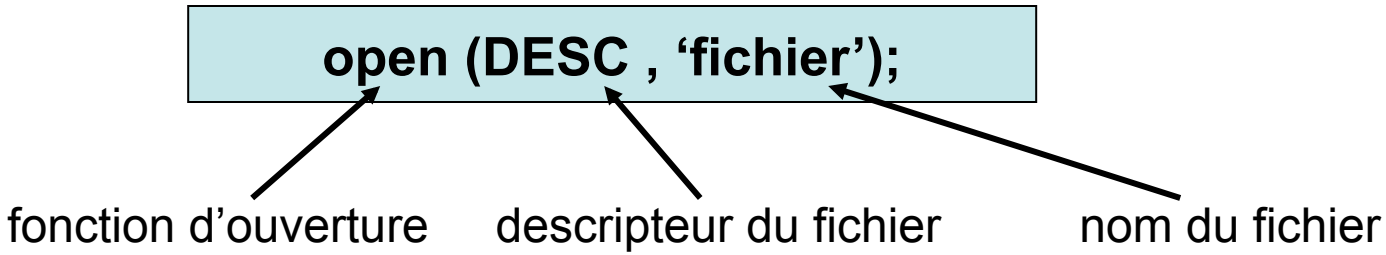
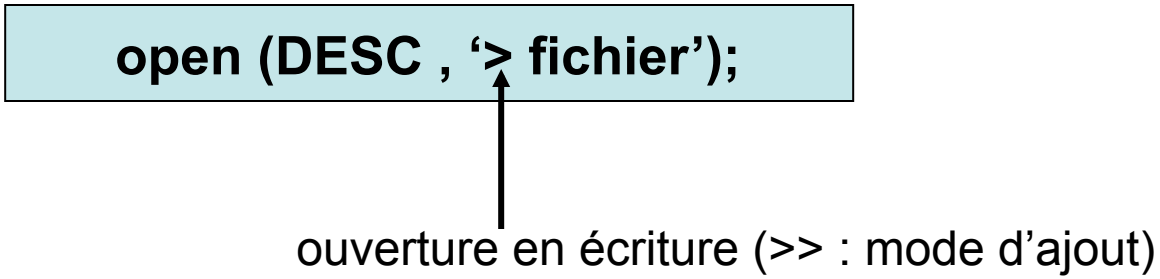
Les fichiers : ouverture

En lecture : la fonction **open** permet d'ouvrir un fichier et prend en paramètre un descripteur (l'objet permettant de manipuler le fichier) et le nom du fichier.

Format :



En écriture : il faut ajouter le symbole '>' devant le nom du fichier. Si le fichier existe, son contenu sera écrasé. En mode d'ajout, on utilise le symbole '>>'. Si le fichier n'existe pas, il sera créé.



Les fichiers : ouverture / fermeture

Gestion d'erreurs : Lorsqu'on ouvre un fichier, il se peut qu'il y ait une erreur. En lecture: le fichier n'existe pas ou ne peut pas être lu. En écriture: le fichier existe, mais on n'a pas le droit d'y écrire.

La fonction **die** permet d'afficher un message et d'arrêter le programme.

```
if ( ! open (DESC , '> fichier') ) {
        die "Problème à l'ouverture : $!";
}
```

```
open (DESC , '> fichier') || die "problème à l'ouverture : $!";
```

La variable **$!** Contient le message d'erreur système. On peut aussi afficher son contenu.

La fermeture du fichier s'effectue avec la fonction **close**.
exemple : close(DESC);

Les fichiers : lecture / écriture

La lecture dans un fichier se fait par ligne.

```
while ($ligne = <DESC>) {
    print $ligne ;
}
```

On peut aussi lire tout le fichier et le placer dans un tableau. À chaque indice du tableau, il y aura une ligne du fichier.

```
@ligne = <DESC> ;
```

L'écriture dans un fichier est encore plus simple. Il suffit de spécifier le descripteur du fichier à l'utilisation du **print**.

```
print DESC " allo le monde " ;
```

Les fichiers : lecture / écriture (2)

Quelques exemples équivalents de lecture d'un fichier:

```
while (defined($line = <DESC>)) {
        print "I saw $line";
}
```

```
while (<DESC>) {
        print "I saw $_";
}
```

```
while (defined($_ = <DESC>)) {
        print "I saw $_";
}
```

```
foreach (<DESC>) {
        print "I saw $_";
}
```

L’opérateur < >

L’opérateur <> (*diamond operator*) est utilisé en Perl pour effectuer la lecture des fichiers dont les noms sont introduits sur la ligne de commande. Si aucun fichier n’est spécifié sur la ligne de commande, l’entrée se fait de l’entrée standard <STDIN>.

La ligne de commande:

$ ./my_program fred barney betty

Deux versions équivalentes du programme qui lisent les lignes des fichiers fred, barney et betty:

```
while (defined($line = <>)) {
        chomp($line);
        print "It was $line that I saw!\n";
}

while (<>) {
        chomp;
        print "It was $_ that I saw!\n";
}
```

La sortie formatée avec l'opérateur printf

Quelques exemples d'utilisation de printf:

```
$user = "merlyn";
$days_to_die = 3;
printf "Hello, %s; your password expires in %d days!\n", $user, $days_to_die;
# Imprime: Hello, merlyn; your password expires in 3 days!

printf "%g %g %g\n", 5/2, 51/17, 51 ** 17;      # 2.5 3 1.0683e+29

printf "in %d days!\n", 17.85;                  # imprime: in 17 days!

printf "%6d\n", 42;                             # la sortie est comme suit:    42

printf "%2d\n", 2e3 + 1.95;                     # 2001

printf "%10f\n", 6 * 7 + 2/3;                   # la sortie est comme suit: 42.6666667

printf "%10.3f\n", 6 * 7 + 2/3;                 # la sortie est comme suit:      42.667

printf "%10.0f\n", 6 * 7 + 2/3;                 # la sortie est comme suit:          43
```

Les identifiants des fichiers

Les identifiants réservés par Perl:

STDIN, STDOUT, STDERR, DATA, ARGV et ARGVOUT.

Quelques exemples d'ouverture de fichiers:

```
open CONFIG, "dino";        # ouverture du fichier dino
open CONFIG, "<dino";       # ouverture du fichier dino en lecture
open BEDROCK, ">fred";      # ouverture du fichier dino en écriture
open LOG, ">>logfile";      # ouverture du fichier dino en ajout
```

Quelques exemples d'ouverture de fichiers :

```
open CONFIG, "<", "dino";
open BEDROCK, ">", $file_name;
open LOG, ">>", &logfile_name( );
```

Fermeture de fichiers:

```
close BEDROCK;
```

Les fonctions die et warn

La fonction **die** permet d'arrêter le programme et d'afficher un message d'erreur (celui d'utilisateur):

```
if ( ! open LOG, ">>logfile") {
        die "Cannot create logfile !";
 }
```

(ou celui du système d'exploitation):

```
if ( ! open LOG, ">>logfile") {
        die "$!";
 }
```

La fonction **warn** permet aussi d'afficher un message d'erreur, mais n'arrête pas l'exécution du programme:

```
if ( ! open LOG, ">>logfile") {
        warn "Cannot create logfile !";
 }
```

Le changement de la sortie par défaut

Un identifiant de fichier peut être utilisé avec la commande print:

```
print LOG "Captain's log, stardate 3.14159\n";          # la sortie ira au fichier LOG
printf STDERR "%d percent complete.\n", $done/$total * 100;
```

Cette syntaxe est aussi correcte (mais pas recommandée):

```
printf (STDERR "%d percent complete.\n", $done/$total * 100);
printf STDERR ("%d percent complete.\n", $done/$total * 100);
```

Utilisation d'un autre identifiant de sortie par défaut au lieu de STDOUT:

```
select BEDROCK; # la commande select permet de changer l'identifiant par défaut
print "I hope Mr. Slate doesn't find out about this.\n";
print "Wilma!\n";
```

Les fichiers : exemple

```
 1 #!/usr/bin/perl
 2
 3 use strict; use warnings;
 4
 5 open (IN,'input.txt') || die " Erreur lors de l'ouverture du fichier : $!";
 6 open (OUT,'>output.txt') || die "Erreur lors de l'ouverture du fichier : $!";
 7
 8 while (my $ligne = <IN>){
 9     print OUT $ligne;
10 }
11
12 close(IN);
13 close(OUT);
```

Ce programme effectue la copie du fichier *input.txt* vers le fichier *output.txt*.

Les variables spéciales et tableaux spéciaux

Petit récapitulatif des variables spéciales: ce sont les variables sous la forme $c (où c est un caractère non-alphabétique). Les tableaux spéciaux commencent par le caractère @ ou %.

Variable	*Description*
$_	variable par défaut courante (par exemple dans foreach)
$!	dernière erreur (est utilisée pour la détection d'erreurs)
$0	nom du programme
@_	contient les paramètres passés à une fonction
@ARGV	tableau des arguments passés au programme
%ENV	tableau des variables d'environnement
@INC	tous les répertoires Perl contenant des librairies

La ligne de commande

Comme en Java, et dans la plupart des langages de programmation, on peut passer des paramètres au programme sur la ligne de commande. En Perl, ces paramètres sont lus dans la variable spéciale @ARGV.

Exemple : (soit le programme copie.pl)

```
 1 #!/usr/bin/perl
 2
 3 use strict; use warnings;
 4
 5 open (IN,$ARGV[0]) || die " Erreur lors de l'ouverture du fichier $ARGV[0] : $!";
 6 open (OUT,"> $ARGV[1]") || die "Erreur lors de l'ouverture du fichier $ARGV[1] : $!";
 7
 8 while (my $ligne = <IN>){
 9     print OUT $ligne;
10 }
11
12 close(IN);
13 close(OUT);
```

Appel : > perl copie.pl input.txt output.txt

Les fonctions système

Nom	*Description*	*Exemple*
print	Affichage d'une chaîne	print 'bonjour le monde';
die	Arrêt du programme en affichant un message	if ($fruit -ne 'fraise') { die ' dommage !'; }
exit	Arrêt du programme.	if ($fruit ne 'fraise') { exit; }
system	Exécute un programme du système	system 'mkdir inf7212';
sleep n	Le programme « dort » pendant n secondes.	sleep 100;

Les fonctions mathématiques

Nom	*Description*	*Exemple*
sin	sinus	$res = sin (0) ;
cos	cosinus	$res = cos ($val) ;
log	logarithme	$res = log (20) ;
int	valeur entière	$res = int (10.2) ; => 10
sqrt	racine carrée	$res = sqrt (20) ;
rand(n)	entiers pseudo-aléatoire	$res = rand(100) ;
abs	valeur absolue	$res = abs (-10) ; => 10

Exercices (4)

1) Écrire un programme Perl permettant d'associer un prix à chaque fruit (en utilisant un tableau associatif). Les prix doivent être saisis au clavier. Puis utilisez une boucle pour trouver la pomme et afficher son prix.

2) Écrire un programme Perl permettant d'effectuer un tri bulle (voir le pseudo-code sur le diapositif suivant).

Coder ces programmes à la démo !!

Solution (exercice 1)

```
#!/usr/bin/perl

use strict;
use warnings;

my %fruits = ('pommes'=>0 , 'poires'=>0);
my $prix;

foreach my $elt (keys % fruits) {
        print STDOUT "\nSaisir le prix des $elt : ";
        $prix = <STDIN>;
        chomp ($prix);
        $fruits{$elt} = $prix;
}

foreach my $elt (keys %fruits) {
        print STDOUT "\n$elt -> " . $fruits{$elt};
}
```

Exercice (tri bulle)

Écrire une fonction permettant d'effectuer le tri bulle d'entiers.

Écrire un programme permettant de lire sur la ligne de commande les valeurs à trier, puis d'effectuer le tri.

Pseudo-code de l'algorithme du tri bulle :

```
PRODECURE Tri_bulle (Tableau a[1:n])
        VARIABLE permut : Booleen;
        REPETER
                permut = FAUX
                POUR i VARIANT DE 1 à N-1 FAIRE
                        SI a[i] > a[i+1] ALORS
                                echanger a[i] et a[i+1]
                                permut = VRAI
                        FIN SI
                FIN POUR
        TANT QUE permut = VRAI
FIN PROCEDURE
```