# Travaux pratiques: programmation en Shell

Les exercices sont tous indépendants et peuvent donc être traités dans un ordre quelconque. Pour tous ces exercices, il est important:

- de tester les arguments (nombre, accessibilité, type ...);
- et de rediriger les sorties «inutiles» pour n'offrir à l'utilisateur que les messages émis explicitement par les commandes.

Liste des exercices:


1. Shell
2. Commande which
3. Affichage de la date
4. Fonctions liées au système de fichiers
5. Affichage de fichiers avec une bannière
6. Renommage interactif de fichiers
7. Envoi d'un message
8. Recopie d'une arborescence
9. Jour de la semaine
10. Gestion d'un annuaire
11. Gestion d'un calepin


## 1. Trouver les scripts shell

Écrire un script qui liste les noms de fichiers de type "Shell script" dans une arborescence donnée (par exemple `/usr`). Affichez le nombre de lignes de chaque fichier.

## 2. Commande which

Écrire un script imitant la commande `which`:

- parcourir les répertoires précisés par la variable PATH et afficher le premier chemin contenant la commande donnée en argument,
- Prendre en compte l'option `a` qui affiche tous les chemins possibles; utiliser pour cela la commande interne `getopts`.
- Analyser le script de la commande `which`.

## 3. Affichage de la date

La commande `date` affiche la date en anglais ou de manière abrégée. Écrivez une fonction shell `datefr`qui l'affiche complètement.

Exemple:

```
Lundi 4 décembre 2006 à 14h

# Affichage de la date en francais

datefr () {

set `date`              # recuperation de la date en $1, $2 ...

case $1 in              # conversion du jour
Mon | lun) jour=Lundi;;
Tue | mar) jour=Mardi;;
Wed | mer) jour=Mercredi;;
Thu | jeu) jour=Jeudi;;
Fri | ven) jour=Vendredi;;
Sat | sam) jour=Samedi;;
Sun | dim) jour=Dimanche;;
*) jour="rien";;
esac

case $2 in              # conversion du mois
Jan | jan) mois=Janvier;;
Feb | fev) mois=Fevrier;;
Mar | mar) mois=Mars;;
Apr | avr) mois=Avril;;
May | mai) mois=Mai;;
Jun | jui) mois=Juin;;
Jul | jui) mois=Juillet;;
Aug | aou) mois=Aout;;
Sep | sep) mois=Septembre;;
Oct | oct) mois=Octobre;;
Nov | nov) mois=Novembre;;
Dec | dev) mois=Decembre;;
*) mois="rien";;
esac

quant=$3                # sauvegarde du jour du mois
an=$6
IFS=:                   # modification des separateurs ( : )
set $4                  # recuperation de l'heure
heure=$1H$2             # concatenation heures et minutes

# affichage du resultat

echo "$jour $quant $mois $an a $heure "
}
```

# 4. Fonctions liées au système de fichiers

Créer un fichier contenant la déclaration de fonctions offrant une "syntaxe" à la MS_DOS telles que

- `dir [arg...]`: affichage du contenu d'un répertoire (0 ou plusieurs arguments)
- `cwd [arg]`: déplacement dans l'arborescence avec un prompt indiquant le répertoire courant (0 ou 1 argument)
- `ren arg1 arg2`: renommage (2 arguments)
- `del arg [...]`: suppression (au moins 1 argument)

- `typ arg [...]`: affichage de fichiers (au moins 1 argument)

Le Shell exécute t-il en premier ces fonctions ou ses propres fonctions internes? ici on a utilisé les noms `cwd` et `typ` au lieu de `cd` et `type` qui sont des fonctions internes pour éviter toute ambiguïté.

```
# fonctions en Shell donnant "l'equivalent" de commandes MS-DOS

dir()                   # fonction dir (affichage d'un repertoire)
{

case $# in
0) rep=`pwd`;;
*) rep=$*;;
esac

for f in $rep
do

if [ -d $f ]                    # est ce un repertoire ?
then
  if [ -r $f -a -x $f ]         # lisible et traversable ?
    then
        ls -la $f
  else
        echo "Le repertoire $f est inaccessible" 1>&2;
  fi

# sinon on teste si l'element existe et est accessible
# en testant le resultat de ls -l

else
  r=`ls -l $f 2>/dev/null`
  if [ "$r" ]
    then
      echo $r
  else
      echo "L'element $f est inaccessible" 1>&2;
  fi
fi
done
}



# fonction cdbis = cd + prompt avec le repertoire courant (nom relatif)
# cd est une fonction shell --> celle-ci est appelee cdbis

cdbis()
{
case $# in
0) cd $HOME;;
1) if [ -d $1 -a -x $1 ]
   then
        cd $1
   else
        echo "$1 est inaccessible"
   fi;;
 *) echo "ERREUR: usage: cd [rep]" 1>&2;
```

```
    exit 1;;
 esac
 ici=`pwd`;
 ici=`basename $ici`;
 PS1="$ici> "

}

# fonction rename (renommage d'un fichier)

rename()
{
case $# in
2) if  mv $1 $2 2>/dev/null
   then :
   else
        echo ERREUR: $1 inaccessible 1>&2;
        exit 1;
   fi;;
*) echo "ERREUR: usage: rename fichier1 fichier2" 1>&2;
   exit 1;;
esac
}



# fonction del (suppression de fichiers)
del()
{
case $# in
0) echo "ERREUR: usage: del fichiers ..."  1>&2; exit 1;;
*) for f in $*
   do
     if rm $f 2>/dev/null
     then :
     else echo "L'element $f est inaccesible"
     fi
   done;;
esac
}



# fonction typ (type est une fonction shell, celle-ci est appelee typ)
type()
{
case $# in
0) echo "ERREUR: usage: typ fichiers ..." 1>&2; exit 1;;
*) for f in $*
   do
        if cat $f 2>/dev/null
        then
        else
           echo "ERREUR: l'element $f est inaccessible" 1>&2;
           exit 1;
        fi
   done;;
esac
}
```

# 5. Affichage de fichiers avec une bannière

La commande `cat f1 f2 f3 ...` affiche les fichiers `f1`, `f2`, `f3`, etc sans séparer la fin de `fi` du début de `fi+1`. Écrivez un script shell qui ajoute au début et à la fin de chaque fichier une bannière comportant par exemple, la date (en français), le nom de l'utilisateur et le nom du fichier (nom absolu ou relatif).

*Indication: utilisez la fonction `datefr` définie précédemment.*

```
# Affichage de fichiers avec pour en-tete :
# la date , le nom d'utilisateur et le nom du fichier

# fonction d'affichage de l'entete
# j, q et m sont des variables positionnees apres set datefr
# USER est une variable de l'environnement
# la commande basename permet de ne garder que le nom relatif

entete()
{
echo "
------------- $j $q $a $m ----- $USER ----- `basename $f` ------------
 "
}

# test du nombre de parametres

if [ $# -eq 0 ]
then
  echo "ERREUR: usage : `basename $0` fichier1 [...]" 1>&2;
  exit 1
fi

# sauvegarde des arguments

file=$*

# recuperation de la date en francais
if ! type datefr > /dev/null
  then
    echo "ERREUR: probleme avec datefr" 1>&2;
    exit 1
fi

set `datefr`
j=$2;q=$3;a=$4;m=$6

# boucle d'affichage des fichiers

for f in $file
do

# test d'accessibilite des fichiers
if [ ! -f $f -o ! -r $f ]
   then
   echo "ERREUR: le fichier $f est inaccessible " 1>&2;
   exit 1
fi
```

```
entete
cat $f
entete

done
```

Pour utiliser la fonction `datefr`, plusieurs solutions:

- exécuter `. fonctionDate.script` (script contenant la définition de la fonction) puis exporter la fonction: `export -f datefr`;
- exécuter `. fonctionDate.script` directement dans le script `mycat.script`;
- exécuter `mycat.script` dans le même shell: `. mycat.script`. (Attention alors aux `exit 1`!)

# 6. Renommage interactif de fichiers

Ecrire une commande qui renomme les entrées d'un répertoire. Pour chacune des entrées du répertoire, l'utilisateur doit saisir un nouveau nom s'il souhaite renommer l'entrée; l'entrée est alors renommée. Cette commande doit:

- afficher chaque entrée du répertoire courant (si la commande est appelée sans argument) ou d'un répertoire passé en argument (en vérifiant que celui-ci existe);
- lire la réponse de l'utilisateur : `RETURN` signifiera «pas de renommage du fichier correspondant» et toute autre réponse sera utilisée comme nouveau nom du fichier;
- tester l'existence d'une entrée ayant le même nom que la réponse.

La commande `mv f1 f2` écrase le fichier `f2` si celui-ci existe. Dans ce cas, soit affichez un message et passez au fichier suivant, soit demandez un nouveau nom, soit demandez confirmation.

```
#!/bin/bash
# rename.script
# renommage interactif des fichiers d'un repertoire
# Syntaxe : rename [repertoire]
#

# test du nombre d'arguments :
# 0 argument : utilisation du repertoire courant
# 1 argument : nom du repertoire
# n arguments : erreur de syntaxe

case $# in
0) Dir=`pwd`;;
1) if [ -d $1 ]                # test si l'argument est un repertoire
   then
     Dir=$1
   else
     echo "ERREUR: $1 n'est par un repertoire"  1>&2;
     exit 1
   fi;;
*) echo "ERREUR: usage : $0 [repertoire]" 1>&2;
   exit 1 ;;
```

```
esac

# test d'accessibilite du repertoire
if [ ! -w $Dir ]
then
    echo "ERREUR: acces interdit sur le repertoire $Dir" 1>&2;
    exit 1
fi

# boucle de lecture des entrees

for old in `ls $Dir`
do
 echo  "$old    : nouveau nom ou RETURN : "    # question
 read new                                      # lecture reponse

 case $new in                                  # test reponse
 "") continue;;
 *)  if [ -f $new -o -d $new ]                 # test existance reponse
     then
       echo "ATTENTION: $new existe deja, renommage refuse"  # ici refus
simple
     else
       mv $Dir/$old $Dir/$new                          # renommage
       echo $Dir/$old renomme en $Dir/$new
     fi;;
 esac
done
```

# 7. Envoi d'un message

Écrire une procédure `envoi nom fich` qui envoie le contenu du fichier `fich` à l'utilisateur `nom` soit par `write` s'il est connecté et accepte les messages soit par mail.

La commande devra tester la syntaxe d'appel (2 arguments), l'existence du fichier `fich`, la déclaration de l'utilisateur `nom` dans le fichier `/etc/passwd` (une entrée commençant par `nom:`).

```
#!/bin/bash
# procedure envoi
# Syntaxe : envoi nom fich
#
# Le fichier fich est envoye a l'utilisateur nom soit par write s'il est
#  connecte et accepte les messages, soit par mail

# test de la syntaxe d'appel
if [ $# -ne 2 ]
  then echo "ERREUR: Usage : $0 nom fich" 1>&2;
  exit 1
fi

# test si l'utilisateur est bien declare

if ! grep "^$1:" /etc/passwd >/dev/null
 then
  echo "ERREUR: utilisateur $1 non enregistre dans /etc/passwd" 1>&2;
  exit 1
fi
```

```
echo "Utilisateur trouve"

if [ ! -f $2 ]                          # test si le fichier fich existe
 then
  echo "ERREUR: fichier $2 n'existe pas" 1>&2;
  exit 1
fi

echo "fichier existe"

if ! who|grep "^$1 " >/dev/null       #test si l'utilisateur est connecte
then
  echo "ERREUR: utilisateur $1 non connecte" 1>&2;
  exit 1
fi

echo "utilisateur connecte"

echo "essai write"


if write $1 <$2 >/dev/null            # envoi par write
 then
 echo "SUCCES: write OK"
 exit 0                # Ok l'envoi s'est bien deroule
fi

echo "essai mail"


if mail $1 <$2                         # envoi par mail
 then
   echo "SUCCES: mail OK"
   exit 0
fi
```

# 8. Recopie d'une arborescence

Écrire une procédure `svgdir r1 r2` qui effectue la copie du répertoire `r1` (qui doit exister) sur un répertoire `r2` (qui ne doit pas exister) en reconstruisant la même arborescence.

Cet exercice fait intervenir la récursivité. Deux méthodes sont possibles, soit se déplacer dans l'arborescence à copier, soit ne pas se déplacer.

Sans déplacement:

```
#!/bin/bash
# svgdir1.script
# svgdir : Sauvegarde d'un répertoire avec reconstruction
#          de la même arborescence
# Sans déplacement dans l'arborescence

# test de la syntaxe
# 2 arguments : répertoire à sauvegarder et répertoire d'arrivée

case $# in
```

```
2) orig=$1;
   dest=$2;;
*) echo "ERREUR: usage : $0 rep_origine rep_dest" 1>&2;
   exit 1;;
esac

case $0 in
/*)     svgdir=$0;;                  # deja un nom absolu
*)      svgdir=`pwd`/$0;;            # nom absolu de la commande
esac

# test : existence du répertoire à sauvegarder

if [ ! -d $orig ]
then
  echo "ERREUR: $orig n'est pas un repertoire " 1>&2;
  exit 1;
fi

if [ ! -r $orig ]
then
  echo "ERREUR: $orig n'est pas lisible " 1>&2;
  exit 1
fi

if [ ! -x $orig ]
then
  echo "ERREUR: $orig n'est pas traversable " 1>&2;
  exit 1
fi

# test : existence du repertoire d'arrivee
if [ -d $dest ]
then
  echo "ERREUR: $dest existe deja " 1>&2;
  exit 1
fi

if [ ! -w `dirname $dest` ]
then
  echo "ERREUR : creation de $dest impossible " 1>&2;
  exit 1;
fi

mkdir $dest                    # création du répertoire d'arrivée

# boucle de lecture

for i in `ls -a $orig`
do
  if [ $i = '.' -o $i = '..' ]
  then
  continue     # ignorer les entrées . et ..
  fi

  if [ -d $orig/$i ]   # test s'il s'agit d'un répertoire
  then
    $svgdir $orig/$i $dest/$i # si oui recursivité (nom complet)
  elif [ -r $orig/$i ]
    then
         cp $orig/$i $dest     # sinon copie simple de fichiers
```

```
    else
         echo "$orig/$i n'est pas lisible"
 fi
done
```

## Avec déplacement:

```
#!/bin/bash
# svgdir2.script
# svgdir : Sauvegarde d'un répertoire avec reconstruction
#          de la même arborescence

# Avec déplacement dans l'arborescence (~domy/SHELL/svgdir_v1)

# test de la syntaxe
# 2 arguments : répertoire à sauvegarder et répertoire d'arrivée

case $# in
2) orig=$1;
   dest=$2;;
*) echo "ERREUR: usage : $0 rep_origine rep_dest" 1>&2;
   exit 1
esac

case $0 in
/*)     svgdir=$0;;                    # deja un nom absolu
*)      svgdir=`pwd`/$0;;             # nom absolu de la commande
esac



# test : existence du répertoire à sauvegarder

if [ ! -d `basename $orig` ]
then
  echo "ERREUR: $orig n'est pas un repertoire " 1>&2;
  exit 1
fi

if [ ! -r `basename $orig` ]
then
  echo "ERREUR: $orig n'est pas lisible " 1>&2;
  exit 1
fi
if [ ! -x `basename $orig` ]
then
  echo "ERREUR: $orig n'est pas traversable " 1>&2;
  exit 1
fi

# test : existence du répertoire d'arrivée
if [ -d $dest ]
then
  echo "ERREUR: $dest existe deja " 1>&2;
  exit 1
fi

if [ ! -w `dirname $dest` ]
then
  echo "ERREUR: creation de $dest impossible " 1>&2;
  exit 1
```

```
fi

# création du répertoire d'arrivée

mkdir $dest

cd $orig
# boucle de lecture

for i in `ls -a`
do
  if [ $i = '.' -o $i = '..' ]
  then
  continue     # ignorer les entrées . et ..
  fi
  if [ -d $i ]  # test s'il s'agit d'un répertoire
  then
    $svgdir $orig/$i $dest/$i  # si oui recursivité (nom complet)
  elif [ -r $i ]
    then
         cp $i $dest           # sinon copie simple de fichiers
    else
         echo "$i n'est pas lisible" 1>&2;
 fi
done
```

# 9. Jour de la semaine

Écrire une procédure qui affiche à quel jour de la semaine correspond une date donnée.

Exemple : le 14 Juillet 1789 est un mardi.

La date peut être passée comme argument à la commande ou lue au clavier sous une forme à déterminer (14/7/1789 ou 14 7 1789 ou 14 Juillet 1789 ou ...). Vérifiez la validité de la date.

*Indication : la commande* `cal x y` *affiche le calendrier du mois* `x` *de l'année* `y`*.*

```
#!/bin/bash
# queljour.script
# recherche du jour de la semaine pour une date donnee

case $# in
        3) ;;
        *) echo "ERREUR: usage : `basename $0` jour mois annee" 1>&2;
           exit 1;;
esac
# sauvegarde des arguments d'appel
jour=$1 ; mois=$2 ; an=$3

if ! cal $mois $an |tail +2|grep -e $jour >/dev/null
  then
       echo "ERREUR: date incorrecte ($jour/$mois/$an)" 1>&2;
       exit 1;
fi

# La commande cal a en sortie le format suivant :
#       ligne 1 : mois et annee
#       ligne 2 : jours (S M Tu W Th F S : Dim Lun Mar Mer Jeu Ven Sam)
```

```
# Ces 2 lignes seront elimines et le test portera sur un jour avec 2
chiffres
# sachant que le 1er est le meme jour que le 8 et le 15, le 5 que le 12 ...

# conversion de 1 et 2 en 15 et 16 et de 3,4 .. 9 en 10,11 .. 16
# pour garantir l'unicite de la recherche (sinon 6 se retouve dans 16 et
26...)

case $jour in
        0) echo "ERREUR: Date incorrecte" 1>&2;
           exit 1;;
        1|2) let j=$jour+14;;
        3|4|5|6|7|8|9) let j=$jour+7;;
        *) j=$jour;;
esac

# recuperation de la semaine concernee en eliminant les 3 premieres lignes
# puis de la position du jour recherche

set `cal $2 $3 | tail +2 | grep "$j"`
# traduction de la position en un jour de la semaine

case $j in
        $1) JOUR=Dimanche;;
        $2) JOUR=Lundi;;
        $3) JOUR=Mardi;;
        $4) JOUR=Mercredi;;
        $5) JOUR=Jeudi;;
        $6) JOUR=Vendredi;;
        $7) JOUR=Samedi;;
esac
echo "Le $jour $mois $an est un $JOUR"
```

# 10. Gestion d'un annuaire

On se propose d'écrire un script shell pour gérer un annuaire. Cet annuaire comporte une série de lignes, chacune composée de 5 champs : nom, prénom, numéro de téléphone, bureau et service. Les champs sont séparés par le caractère deux-points comme le montre l'exemple ci-après :

- annuaire.txt

Ce script shell est appelé avec une seule option pour réaliser chacune des fonctions ci-après :

- afficher l'annuaire trié par service et par nom (-t)
- afficher l'annuaire trié sur le numéro de téléphone (-T)
- afficher la liste des noms des inscrits (-M)
- afficher le dernier inscrit (-d)
- afficher la liste des inscrits sous la forme Nom.Prénom (-m)
- rechercher un inscrit à partir de son nom (-g bac) ou d'une partie seulement (-g bou) et sans distinction des majuscules/minuscules
- ajouter un nouvel inscrit (-a)
- créer un fichier par service (-e)
- supprimer un inscrit (-s Bac)

- lister le personnel d'un bâtiment (-b X). Le bâtiment est indiqué par la première lettre du bureau.
- etc (selon votre imagination ou vos besoins).

En outre, le programme doit respecter les consignes ci-après :

- tester la syntaxe d'appel : il faut au moins un argument : le nom de l'annuaire et au maximum une option
- tester si l'annuaire existe et est accessible
- sans option, le script shell affiche simplement l'annuaire trié par nom
- pour l'ajout d'un inscrit, boucler en lecture clavier pour saisir successivement chacun des champs et demander confirmation avant de l'enregistrer
- pour la suppression, l'argument doit représenter exactement le premier champ
- être protégé contre les signaux TERM (N° 15) et INTR (N° 2 et touche CTRL C).

```
#!/bin/bash
# Auteurs: Dominique Bouillet, Daniel Millot

refus()
{
echo Ne pas interrompre le programme...
}


ajout()
{
read -p"Nom : " nom
read -p"Prenom : " prenom
read -p"Numero de telephone : " tel
read -p"Bureau : " buro
read -p"Service : " serv

echo "$nom:$prenom:$tel:$buro:$serv"
echo "Confirmez-vous ? (o/n)"
read reponse
case "$reponse" in
o|oui)  echo "$nom:$prenom:$tel:$buro:$serv" >>$Annuaire
        echo Utilisateur $nom enregistré;;
n|non)  echo "Information NON enregistrée"
        ;;
esac

}

eclat()
{
for f in $(cut -d: -f5 $Annuaire | sort -u)
do
grep $f $Annuaire >>$f
done
}

# Debut du programme

trap "refus" 2 3 15

if [ $# -lt 1 ]
then    echo "Syntaxe : $(basename $0) [-opt] Nom_Annuaire" 1>&2;
```

```
        exit 1
fi

eval Annuaire=\${$#}
if [[ ! -e "$Annuaire" || ! -r "$Annuaire" ]]
then    echo "\"$Annuaire\" devrait être le nom d'un fichier annuaire
lisible"
        exit
fi

if [ $# -eq 1 ]
then    sort "$Annuaire"
else
case $1 in

-t)     sort -t: -k 5 -k 1 "$Annuaire";;
-T)     sort -t: -k 3n "$Annuaire";;
-M)     cut -d: -f1 "$Annuaire";;
-d)     tail -1 "$Annuaire";;
-m)     tr ':' '.' <"$Annuaire" | cut -d. -f1,2;;
-a)     ajout;;
-g)     if [ $# -ne 3 ]
        then    echo "Syntaxe : $(basename $0) -g nom Nom_Annuaire"  1>&2;
                exit 1
        else
                grep -i "^[^:]*$2[^:]*:" "$Annuaire"
        fi;;
-e)     eclat;;
-s)     if [ $# -ne 3 ]
        then    echo "Syntaxe : $(basename $0) -s Nom Nom_Annuaire"  1>&2;
                exit 1
        else
                if grep "^$2" $Annuaire >/dev/null 2>&1
                then
                grep -v "^$2" $Annuaire >/tmp/ann.$$
                mv $Annuaire ${Annuaire}.old
                mv /tmp/ann.$$ $Annuaire
                echo $2 supprime
                else
                echo "L'utilisateur $2 n'est pas inscrit"
                fi
        fi;;
-b)     if [ $# -ne 3 ]
        then    echo "Syntaxe : $(basename $0) -b X Nom_Annuaire" 1>&2;
                exit 1
        else
                grep "^[^:]*:[^:]*:[^:]*:$2.*:" $Annuaire | cut -d: -f1|sort
        fi;;
*)      echo "Option ($1) inconnue";exit;;
esac
fi
```

# 11. Gestion d'un calepin

On se propose d'écrire un script shell pour gérer un calepin de notes. Une note est définie comme étant une seule ligne de texte ASCII et les notes sont numérotées de 1 à n.

Ce script shell offre les fonctions suivantes :

- aide en ligne : liste des commandes avec leur syntaxe (h)
- affichage de la note courante (l) ou de la note x (lx)
- affichage de la note suivante (+ ou Return)
- affichage de la note précédente (-)
- destruction de la note courante (d) ou de la note x (dx)
- ajout d'une note en fin de calepin (a)
- affichage du nombre de notes du calepin (n)
- sortie du programme (q ou Ctrl D)
- affichage du calepin complet (p)
- traitement des notes par contenu (l/exp/ ou d/exp/)
- traitement des notes par intervalles (lx-y ou dx-y)
- etc (selon vos désirs)

Conventions :

- les codes ci-dessus (entre parenthèses) sont des exemples possibles
- les valeurs x et y correspondent à des nombres entiers
- la valeur /exp/ correspond à une expression régulière

Ecrire le programme qui doit :

- tester la syntaxe d'appel : 0 paramètre pour utiliser le nom d'un calepin par défaut ou 1 paramètre : le nom du calepin
- créer un calepin vide ou tester si le calepin est accessible
- afficher soit le nom du calepin créé, soit la dernière note
- afficher un prompt incluant le numéro de note courante
- boucler sur la lecture du clavier pour lire les requêtes de l'utilisateur
- tester la réponse et effectuer le traitement demandé
- être protégé contre les codes intr et quit

## Indications :

- définir une note courante : la dernière note affichée
- lorsqu'elle est détruite, la précédente devient la note courante
- écrire chaque traitement demandé dans une fonction
- utiliser des variables communes : nom du calepin, nombre de notes, note courante, réponse, etc...
- pour afficher la note 6, on peut utiliser sed -n 6p calepin
- tester les valeurs limites (début/fin de calepin pour -, +, etc...)
- écrire une fonction vide qui teste si le calepin est vide et répond vrai ou faux Utiliser son résultat pour conditionner l'appel d'une autre fonction
- écrire les fonctions progressivement : commencer par nombre, puis ajouter. Pour liste et detruire, dans un premier temps, se limiter à la note courante.

---

*Auteur: Dominique Bouillet*
*Modifications: Frédérique Silber-Chaussumier*
Last modified: Thu Nov 19 13:42:08 CET 2009