Exercice 11.1

Écrivez une fonction qui renvoie la somme de cinq nombres fournis en argument.

Exercice 11.2

Écrivez une fonction qui renvoie le nombre de voyelles contenues dans une chaîne de caractères passée en argument. Au passage, notez qu'une fonction a tout à fait le droit d'appeler une autre fonction.

Exercice 11.3

Réécrivez la fonction Trouve, vue précédemment, à l'aide des fonctions Mid et Len (comme quoi, Trouve, à la différence de Mid et Len, n'est pas une fonction indispensable dans un langage).

Exercice 11.4

Ecrivez une fonction qui purge une chaîne d'un caractère, la chaîne comme le caractère étant passés en argument. Si le caractère spécifié ne fait pas partie de la chaîne, celle-ci devra être retournée intacte. Par exemple :

- Purge("Bonjour","o") renverra "Bnjur"
- Purge("J'ai horreur des espaces"," ") renverra "J'aihorreurdesespaces"
- Purge("Moi, je m'en fous", "y") renverra "Moi, je m'en fous"

Exercice 11.5

Même question que précédement, mais cette fois, on doit pouvoir fournir un nombre quelconque de caractères à supprimer en argument. Ainsi, par exemple, si le deuxième argument est "aeiouy", la chaîne passée en premier argument sera purgée de toutes ses voyelles.

Exercice 11.6

Ecrire un traitement qui effectue le tri d'un tableau envoyé en argument (on considère que le code appelant devra également fournir le nombre d'éléments du tableau).

Exercice 11.7

Ecrire un traitement qui informe si un un tableau envoyé en argument est formé ou non d'éléments tous rangés en ordre croissant.

Exercice 11.8

Ecrire un traitement qui inverse le contenu de deux valeurs passées en argument.

Exercice 11.9

reprendre l'exercice 11.6, mais cette fois la procédure comprendra un troisième paramètre, de type booléen. VRAI, celui-ci indiquera que le tri devra être effectué dans l'ordre croissant, FAUX dans l'ordre décroissant.

Exercice 11.10

On va à présent réaliser une application complète, en utilisant une architecture sous forme de sous-procédures et de fonction. Cette application a pour tâche de générer des grilles de Sudoku. Une telle grille est formée de 81 cases (9 x 9), contenant un chiffre entre 1 et 9, et dans laquelle aucune ligne, aucune colonne et aucune "sous-grille" de 3x3, ne contient deux fois le même chiffre.

Pour parvenir à nos fins, on va utiliser une méthode particulièrement barbare et inefficace : la génération aléatoire des 81 valeurs de la grille. On vérifiera alors que la grille satisfait aux critères ; si tel n'est pas le cas... on recommence la génération jusqu'à ce que la grille convienne. En pratique, la probabilité de générer une grille adéquate est si faible que cette méthode prendra sans doute beaucoup de temps, mais passons.

Tout le truc est de piger que vérifier que les neuf cases d'une ligne, d'une colonne, ou d'une sous-grille, sont toutes différentes, c'est en réalité du pareil au même. On va donc factoriser le code procédant à cette vérification sous la forme d'une fonction booléenne **TousDifférents**, à qui on passera un tableau de 9 valeurs

en argument. La fonction renverra donc VRAI si les 9 valeurs du tableau sont toutes différentes, et FAUX sinon.

a. Ecrire la fonction **TousDifferents**

Maintenant, bien que ce ne soit pas indispensable (car ce code n'est pas spécialement répété), on choisit également par pure commodité de confier la génération au hasard de la grille de 81 cases à un module dédié, **RemplitGrille**. (ce module, à qui on passera notre tableau de 81 cases en argument, est forcément une procédure, puisqu'il a pour tâche d'en modifier les 81 valeurs).

b. Ecrire la procédure **RemplitGrille**

Il faut à présent vérifier que l'ensemble des lignes correspond à la condition voulue, à savoir qu'il n'y existe pas de doublons. On réalise donc une fonction, **VerifLignes**, qui va vérifier les neuf lignes de notre grille une par une (en utilisant bien sûr la fonction TousDifférents, déjà écrite) et renvoyer VRAI si toutes les lignes sont correctes, FAUX dans le cas contraire.

c. Ecrire la fonction **Veriflignes**

On procède alors de même avec une fonction chargée de vérifir les colonnes, VérifColonnes.

d. Ecrire la fonction **Verifcolonnes**

...et encore à nouveau, avec cette fois la vrification des neuf "sous-grilles" 3x3.

e. Ecrire la fonction **VerifSousGrilles**

Il ne reste plus qu'à écrire la procédure principale, et l'affaire est dans le sac !

f. Ecrire la procédure principale de l'application