Info113 - TD5

L'objectif du troisième TD est à nouveau d'utiliser des tableaux, des boucles et des structures conditionnelles. Les concepts vus dans ce TD sont :

- les fonctions (encore et toujours !),
- les boucles et structures conditionnelles,
- les tableaux.

Filtres et combinaisons

Exercice 1. Définissez une fonction `une_sur_trois` qui prend en argument un tableau `t`, et qui rend un tableau contenant un élément de `t` sur trois.

```
>>> une_sur_trois([0, 1, 2, 3, 4, 5, 6, 7])
[0, 3, 6]
```

Exercice 2. Définissez une fonction `somme_par_deux` qui prend en argument un tableau de nombres `t`, et qui rend un tableau contenant la somme des éléments de `t` deux à deux.

```
>>> somme_par_deux([0, 1, 2, 3, 4, 5, 6, 7])
[1, 5, 9, 13]
>>> somme_par_deux([2, 3, 5])
[5, 5]
```

Exercice 3. Définissez une fonction `produit_par_trois` qui prend en argument un tableau de nombres `t`, et qui rend un tableau contenant le produit des éléments de `t` trois à trois.

```
>>> produit_par_trois([1, 2, 3, 4, 5, 6])
[6, 120]
>>> produit_par_trois([3, 2, 1, 8])
[6, 8]
```

Exercice 4. Définissez une fonction `plus_petit_que_n` qui prend en argument un tableau de nombres `t` ainsi qu'un nombre `n` et qui rend un tableau contenant uniquement les nombres de `t` qui sont plus petits que `n`.

```
>>> plus_petit_que_n([2, 108, 40, 47, 0], 42)
[2, 40, 0]
```

Exercice 5. Définissez une fonction `duplique` qui prend en argument un tableau et qui retourne une copie de ce tableau dans laquelle chaque élément a été dupliqué.

```
>>> duplique([7, "blabla", 42, 3.14])
[7, 7, "blabla", "blabla", 42, 42, 3.14, 3.14]
```

Exercice 6. Définissez une fonction `supprime_doublons` qui prend en argument un tableau `t` et qui retourne une copie de `t` contenant uniquement la première occurrence de chaque élément.

```
>>> supprime_doublons([0, "blabla", 0, 42, 42, 3.14])
[0, "blabla", 3.14]
```

Encodage des matrices

Dans cette partie, nous allons utiliser des tableaux pour encoder des matrices, c'est à dire des tableaux à deux dimensions. Pour stoquer une matrice ayant n lignes et m colonnes, nous allons utiliser un unique tableau dont :

- la première case contiendra le nombre de lignes (c'est à dire n),
- la seconde case contiendra le nombre de colonnes (c'est à dire m),
- les m cases suivantes contiendront la première ligne de la matrice,
- les m cases suivantes contiendront la seconde ligne de la matrice,
- et ainsi de suite jusqu'à la n-ième ligne.

Exercice 7. Définissez une fonction `nouvelle_matrice` qui prend en argument deux entiers `n` et `m`, et qui retourne un tableau contenant la représentation d'une matrice à n lignes et m colonnes, contenant uniquement des zéros.

Exercice 8. Définissez une fonction `largeur_matrice`, puis une fonction `hauteur_matrice`, qui prennent en argument une matrice `M` (c'est à dire un tableau représentant cette matrice) et qui retournent sa largeur (nombre de colonnes) et sa hauteur (nombre de lignes) respectivement.

Exercice 9. Définissez une fonction `element_matrice` qui prend en argument une matrice `M` ainsi que deux entier `x` et `y`, et qui retourne la valeur contenue dans la case de `M` située à l'intersection de la `x`-ième colonne et de la `y`-ième ligne.

Exercice 10. Définissez une procédure (c'est à dire une fonction qui ne retourne pas de valeurs) `change_valeur_matrice` qui prend en argument une matrice `M`, deux entiers `x` et `y` donnant la position d'une case de la matrice et un nombre `v` qui contient la nouvelle valeur que devra avoir cette case.

Exercice 11. Définissez une procédure `affiche_matrice` prenant en argument une matrice et l'affichant dans le terminal, ligne par ligne, en utilisant `print`. Vous aurez besoin de la fonction `str` pour transformer les nombres en chaîne de caractères.

Exercice 12. Définissez une fonction `matrice_id` prenant en argument un entier `n` et retournant la matrice identité de taille `n` (c'est à dire la matrice carré de taille `n` contenant des `1` sur la diagonale et des `0` ailleurs).

Exercice 13. Définissez une fonction `copie_matrice` prenant en argument une matrice `M` et en retournant une copie. Pourquoi une telle fonction est-elle nécessaire ?

Exercice 14. Définissez une fonction `produit_matrice_scalaire` prenant en argument une matrice `M` et un nombre `s` et retournant une nouvelle matrice contenant la produit de `M` et `s` (c'est à dire une copie de la `M` dont chaque case a été multipliée par `s`).

Exercice 15. Définissez une fonction `somme_matrices` prenant en argument deux matrices `A` et `B` de même taille, et retournant une matrice contenant la somme de `A` et de `B` (c'est à dire leur somme élément à élément).

Exercice 16. Définissez une fonction `transpose_matrice` qui prend en argument une matrice `M`, et qui retourne une nouvelle matrice contenant sa transposée (c'est à dire la matrice `M` des les ligne et les colonnes ont été inversées).

Questions plus complexes

Exercice 17. Rappelez la définition du produit matriciel, puis définissez une fonction `produit_matrices` qui prend en argument deux matrices `A` et `B`, et retourne une nouvelle matrice contenant le produit de `A` et `B`.

Exercice 18. Rappelez la définition du déterminant d'une matrice, puis définissez une fonction `determinant` qui prend en argument une matrice `M` et qui en calcule le déterminant.