

## TP n° 5-6

**Consignes** les exercices ou questions marqués d'un ★ devront être rédigés sur papier (afin de se préparer aux épreuves écrites du partiel et de l'examen). En particulier, il est recommandé d'être dans les mêmes conditions qu'en examen : pas de document ni de calculatrice. Les questions marquées d'un ◇ sont des questions supplémentaires permettant d'aller plus loin (et ne seront pas forcément corrigées en TP). Tous les TPs se font sous Linux.

### Exercices

Cette feuille mélange les concepts de deux cours :

- le cours 5 sur les textes
- le cours 6 sur les fonctions

Il faut aussi quelques rappels sur les tableaux et utilise quelques exemples de partage mémoire vu au cours 7 pour l'exercice 1.

#### Exercice 0

Ouvrir un terminal :

- créer un répertoire TP5 à l'intérieur du répertoire **IntroInfo**
- se placer à l'intérieur du répertoire TP5
- (éventuellement refaire les manipulations du TP3 pour l'utilisation de Python)

On suppose pour les autres exercices que le répertoire TP5 est le répertoire courant.

#### Exercice 2

1. ★ Pour chacun des programmes Python ci-dessous, dire ce qu'ils affichent.

(a) 

```
1 def f (x):
2     print(x)
3
4 f(42)
```

(b) 

```
1 def f (x):
2     x = 43
3
4 y = 42
5 f(y)
6 print(y)
```

(c) 

```
1 def f(x):
2     return x + 10
3
4 def g (y):
5     z = y + 1
6     print(f(z))
7
8 g(42)
```

(d) 

```
1 y = 19
2 def f (x):
3     print(y)
4
5 f(42)
```

(e) 

```
1 y = 19
2 def f (x):
3     y = 17
4     print(y)
5
6 f(42)
7 print(y)
```

(f) 

```
1 y = 19
2 def f(x):
3     global y
4     y = 17
5     print(y)
6
7 f(42)
8 print(y)
```

```

1 y = [ 0, 0, 0 ]
2 def f (x):
3     x[0] = 42
(g) 4
5 f(y)
6 print(y)

```

```

1 y = [ [ 0, 0, 0 ] ]
2 z = y + y
(h) 3 z[0][0] = 42
4 print(y)

```

```

1 y = [ 0 ]
2 def f(x):
3     x = [ 1 ]
(i) 4 f(y)
5 print(y)

```

### Réponse:

- (a) 42
- (b) 42. Attention, ici **x** est un paramètre de **f** (et se comporte comme un variable locale). Le modifier ne modifie pas le **y** à l'extérieur (appel par valeur).
- (c) 53
- (d) 19
- (e) 17 puis 19. Le **y** ligne 3 est une variable locale, qui masque la variable globale de la ligne 1. Le **y** de la ligne 7 fait référence à la variable globale de la ligne 1.
- (f) 17 puis 17. L'instruction **global y** de la ligne 3 indique que dans la fonction **f**, toutes les occurrences de la variable **y** font référence à la variable globale définie à la ligne 1. L'affectation ligne 7 modifie cette variable globale.
- (g) affiche [ 42, 0, 0 ]. Attention, ici on n'a pas modifié **y** (dont pas de contradiction avec la question (b)), **y** est toujours la même référence vers le tableau de 3 cases. Mais le **contenu** du tableau a lui été modifié
- (h) [[42, 0, 0], [42, 0, 0]]. C'est une variante de l'exemple du cours. Le tableau **z** est un tableau différent de **y**, mais les cases 0 et 1 ont le même contenu, c'est à dire la même adresse vers le même tableau interne de 3 cases.
- (i) [ 0 ]. C'est ici le même exemple que pour (b). Modifier le paramètre **x** n'a aucune incidence sur **y**.

### Exercice 3

1. Télécharger le fichier **cdm.txt** depuis la page du cours. L'enregistrer dans le répertoire TP5 à côté des fichiers Python de ce TP. Écrire un programme Python effectuant les actions suivantes :
  - ouverture d'un fichier **cdm.txt** en lecture (celui utilisé lors du TP2)
  - lecture du fichier. Dans ce fichier, chaque ligne contient deux noms de pays et une lettre séparés par un caractère « : ».
  - écriture dans un fichier **cdm2.txt** du premier Pays de chaque ligne. Le fichier doit être créé s'il n'existe pas et écrasé s'il existe.

On ne demande pas de gérer les erreurs. Par exemple, comme le fichier **cdm.txt** contient :

```

BRESIL:CAMEROUN:A
BRESIL:CROATIE:A
BRESIL:MEXIQUE:A
CAMEROUN:CROATIE:A
CAMEROUN:MEXIQUE:A
CROATIE:MEXIQUE:A
AUSTRALIE:CHILI:B
...

```

le fichier **cdm2.txt** doit contenir :

```

BRESIL
BRESIL
BRESIL
CAMEROUN
CAMEROUN
CROATIE
AUSTRALIE
...

```

**Rappel** on rappelle les fonctions utiles du cours :

- `open(chemin, mode)` qui permet d'obtenir un descripteur de fichier à partir du chemin
- `f.readlines()` renvoie le tableau des lignes du fichier
- `f.writelines(tab)` écrit un tableau de lignes dans un fichier. Chaque élément du tableau est une chaîne de caractères qui doit se terminer par « `\n` »
- `s.split(sep)` qui découpe la chaîne `s` selon le séparateur `sep`.

## Exercice 4

1. Écrire une fonction `somme_entre(tab, a, b)` qui prend en argument un tableau d'entiers `tab` et qui renvoie la somme de tous les entiers du tableau compris entre `a` et `b` (au sens large).
2. Écrire au moins 5 tests de votre fonction dont le tableau vide et aucune valeur dans l'intervalle (dans ces deux cas, votre fonction doit renvoyer 0).

## Exercice 5

Le but de cet exercice est de programmer une fonctionnalité relativement avancée (voire la fonction finale en question 8), en utilisant des fonctions auxiliaires.

1. Écrire une fonction `bissextile(a)` qui renvoie `True` si l'année `a` est bissextile et `False` sinon. Une année est bissextile si elle est divisible par 4 et qu'elle n'est pas divisible par 100 ou alors si elle est divisible par 400.
2. Écrire une fonction `nb_jours_annee(a)` qui renvoie le nombre de jours d'une année (on devra réutiliser la fonction précédente).
3. Écrire une fonction `nb_jours_mois(a, m)` qui renvoie le nombre de jours dans le mois `m` (compris entre 1 et 12) de l'année `a`. Il est suggéré d'utiliser un tableau dans cette fonction, plutôt qu'une suite de `if/elif`.
4. Écrire une fonction `nb_jours_date(a, m, j)` qui calcule combien de jours complets se sont écoulés entre le 01/01/a et le j/m/a. Attention, le jour considéré doit être exclu. Par exemple, si on exécute l'appel `nbjoursdate(2020, 1, 1)`, ce dernier doit renvoyer 0 et l'appel `nbjoursdate(2020, 2, 1)` doit renvoyer 31.
5. Écrire une fonction `nb_jours_entre(a1, m1, j1, a2, m2, j2)` qui calcule le nombre de jours écoulés entre la date j1/m1/a1 et j2/m2/a2. Vous pouvez supposer que les dates sont valides et que j1/m1/a1 est avant j2/m2/a2. La deuxième date est exclue. Par exemple, `nbjoursentre(2020, 1, 1, 2020, 1, 1)` renvoie 0 et `nbjoursentre(2019, 1, 1, 2020, 1, 1)` renvoie 365.
6. Écrire une fonction `ord_date(a1, m1, j1, a2, m2, j2)` qui prend en argument deux dates supposées valides et vérifie que la première est inférieure ou égale à la seconde. On pourra commencer par comparer par année, puis en cas d'égalité comparer par mois, puis en cas d'égalité comparer par jours.
7. Écrire une fonction `verifie_date(a, m, j)` qui renvoie
  - `True` si la date passée est valide, c'est à dire :
    - l'année est supérieure ou égale à 1600 et inférieure ou égale à 2100
    - le mois est compris entre 1 et 12
    - le jour est supérieur ou égal à 1 et inférieur ou égal au nombre de jours dans le mois, pour l'année donnée (réutiliser `nb_jours_mois`)
  - `False` dans les autres cas
8. Écrire une fonction `trouve_intervalle_max(fichier)` qui prend en argument le chemin vers un fichier contenant des lignes de la forme :

```
31/10/1981 31/10/2024
01/08/2000 04/07/2010
25/08/2020 26/09/2021
...
```

Si une ligne n'est pas au bon format ou si l'une des deux dates est invalide, ou si la première date est strictement supérieure à la seconde, la ligne est ignorée. La fonction affiche dans la console le numéro de ligne et le nombre de jours de la ligne contenant le plus long intervalle, ainsi que le nombre de

lignes valides (non ignorées). Créer un fichier texte vous permettant de tester votre fonction. Le fichier doit vous permettre de tester plusieurs parties de votre fonction (plusieurs *chemins d'exécutions*, par exemple plusieurs types d'erreurs).

## Bonus

### Exercice 6 ♦

On souhaite écrire un petit programme permettant de faire du « pixel art ». Plus précisément, le programme doit lire la description d'une icône dans un fichier texte. Ce fichier contient un certain nombre de lignes. Chaque ligne comporte une suite de caractères qui peut être :

- soit un caractère #
- soit un caractère espace
- soit un retour à la ligne (en fin de ligne)

Le programme doit lire un fichier texte et le parcourir caractère par caractère pour dessiner l'icône de haut en bas. Chaque caractère # correspond à un carré noir de 20 pixels de côté. Tester votre programme sur les fichiers `ghost.txt` et `star.txt` fournis sur la page du cours.

Modifier ensuite le programme pour faire de *l'anti-aliasing*. Une méthode simple (même si le résultat n'est pas forcément très joli) est que la couleur d'une case est calculée comme la moyenne pondérée de cette case et des 8 (au plus) cases voisines, avec des coefficients bien choisis. Par exemple si une case est noire et entourée de 8 cases blanches (on rappelle que pour les couleurs, le noir vaut 0 et le blanc 1.0), on pourra calculer :

$$c'_{x,y} = \frac{c_{x,y} + \sum_{\substack{-1 \leq i \leq 1 \\ -1 \leq j \leq 1 \\ i \neq 0 \vee j \neq 0}} \alpha * c_{x+i,y+j}}{1 + 8\alpha}$$

Si  $c_{x,y} = 0$  et que toutes les cases voisines sont blanches (1.0) :

$$c'_{x,y} = \frac{0 + 8 \times \alpha \times 1.0}{1 + 8 \times \alpha}$$

Si on prend  $\alpha = 0.125$ , on obtient  $c'_{x,y} = \frac{8}{9} \approx 0.888$ . On aura donc un gris clair.

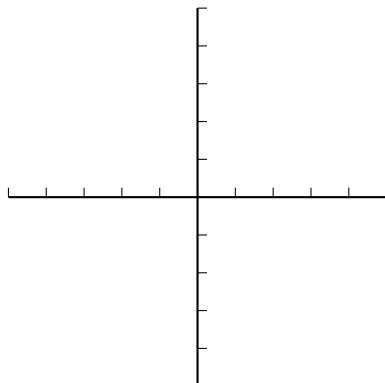
On pourra expérimenter avec différentes valeurs du coefficient  $\alpha$ .

**Réponse:** Voir les fichiers `exo5.py` et `exo5_opt.py`.

### ♦ Exercice 7

On se place dans un fichier où le module `turtle` a été importé. Le but de l'exercice est de tracer la courbe de fonctions mathématiques.

1. Définir une variable globale **F** représentant l'échelle, c'est à dire le nombre de pixels correspondant à une unité. On pourra définir **F** à 100 par exemple.
2. Définir une fonction `dessine_axes()` qui dessine les deux axes du repère orthonormé, avec une graduation de **F/10** pixel de long toutes les unités. On souhaite donc avoir un dessin comme celui-ci :



3. Définir une fonction **f(x)**, qui est la fonction mathématique dont on souhaite tracer le graphe. Cela peut être par exemple  $x^2 - 5x + 3$  ou toute autre fonction de votre choix.
4. Définir une fonction **dessine\_f(a, b, s)** qui dessine le graphe de **f** pour des valeurs de **x** allant entre **a** inclus et **b** exclu, par pas de **s**. Attention, on souhaite que **s** puisse être un nombre flottant, on ne peut donc pas utiliser la fonction **range**.
5. Appeler les deux fonctions pour tracer le repère et le graphe de **f** (ne pas oublier d'appeler **done()** à la suite afin de laisser la fenêtre ouverte).
6. Que peut-il se produire si on définit

```
1  def f (x):  
2      return 1/x
```

et qu'on trace cette fonction ? Modifier **dessine\_f** pour gérer ce cas et ne pas dessiner le graphe pour les valeurs de **x** où **f** n'est pas définie.