

## TP n° 5

**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

#### 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 1

★

1. (Preliminaire) Pour chacun des nombres suivants, écrits en base 2, donner son écriture en base 10.

- (a)  $0000\ 0000_2$
- (b)  $0111\ 1111_2$
- (c)  $1100\ 0000_2$
- (d)  $1101\ 1111_2$
- (e)  $1110\ 0000_2$
- (f)  $1110\ 1111_2$
- (g)  $1000\ 0000_2$
- (h)  $1011\ 1111_2$

#### Réponse:

- (a) 0 (facile)
- (b) On peut poser l'opération, mais aussi se rendre compte que  $0111\ 1111_2 = 1000\ 0000_2 - 0000\ 0001_2 = 2^7 - 1 = 127$
- (c)  $1100\ 0000_2$ . Ici on peut calculer :  $1100\ 0000_2 = 2^7 + 2^6 = 128 + 64 = 192$
- (d)  $1101\ 1111_2$ . On calcule pour l'exemple, même si c'est plus facile de faire le suivant et retirer  
1.  $1101\ 1111_2 = 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 128 + 64 + 16 + 8 + 4 + 2 + 1 = 223$
- (e)  $1110\ 0000_2$ . On peut se rendre compte que c'est le précédent + 1 : 224
- (f)  $1110\ 1111_2$ . On peut se rendre compte que c'est le précédent + 15 :  $224 + 15 = 239$ .
- (g)  $1000\ 0000_2$ . C'est simplement  $2^7 = 128$ .
- (h)  $1011\ 1111_2$ . On peut voir que c'est le nombre (c) - 1, donc 191.

2. Pour chacune des séquences UTF-8 suivantes (écrites en base 10), donner le code Unicode du caractère correspondant, si elle est valide (sinon dire qu'elle est invalide)

- (a) 83

- (b) 195 167
- (c) 172 173
- (d) 226 132 157

**Réponse:**

- (a) 83 est inférieur à 128 (ou dit autrement, le bit de poids fort vaut 0). Le code du caractère est simplement 83.
- (b) Le premier octet est 195. Il est compris entre 192 et 223, il s'agit donc d'un nombre dont la représentation binaire est de la forme  $110x\,xxxx_2$ . Il encode une séquence UTF-8 sur deux octets, de la forme  $110x\,xxxx_2\ 10xx\,xxxx_2$ . On traduit chaque nombre en base 2 :  $195 = 192 + 3 = 1100\,0011_2$  et  $167 = 1010\,0111_2$ . On retire de chaque nombre le préfixe (110 pour le premier octet et 10 pour le second). Le caractère a donc le code :  $0\,001110\,0111_2 = 2^7 + 2^6 + 2^5 + 2^2 + 2 + 1 = 231$
- (c) Le premier octet vaut 172. Il est compris entre 128 et 191, sa représentation en base 2 est de la forme  $10xx\,xxxx_2$ . C'est un octet intermédiaire, il ne peut pas apparaître en début de séquence, la séquence est invalide.
- (d) 226 132 157. 226 est compris entre 224 et 239. Sa représentation en base 2 est de la forme  $1110\,xxxx_2$ , il représente une séquence UTF-8 sur trois octets. On convertit chaque octet en base 2 :  $1110\,0010_2\ 1000\,0100_2\ 1001\,1101_2$ . On retire les préfixes et on réassemble le tout :  $0010\,0001\,0001\,1101_2 = 2^{13} + 2^8 + 2^4 + 2^3 + 2^2 + 1 = 8477$

3. Lancer l'interpréteur Python dans un terminal et utiliser la fonction `chr(n)` pour déterminer à quels caractères correspondent les codes valides.

**Réponse:** On trouve  $83 \rightarrow \text{S}$ ,  $231 \rightarrow \zeta$ ,  $8477 \rightarrow \mathbb{R}$ .

## Exercice 2

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 3

La méthode de chiffrement ROT13 est une méthode simple consistant à substituer toutes les lettres d'un texte par la lettre se trouvant 13 positions plus loin dans l'alphabet. Par exemple `A` devient `N`, `B` devient `O`, ..., `K` devient `X`, `L` devient `Y`, `M` devient `Z`, `N` devient `A`, ...<sup>1</sup>

Par exemple, la chaîne de caractères `"BONJOUR !"` devient `"OBAWBHE !"`. On remarque que l'espace et le point d'exclamation sont inchangés.

- Écrire un programme Python qui demande à l'utilisateur une chaîne de caractères puis la transforme avec la méthode ROT13. On pourra utiliser l'algorithme suivant :
  - lire une chaîne `s`.
  - créer un tableau `t` de même longueur que `s`, contenant par exemple la chaîne vide dans toutes les cases.
  - pour chaque caractère `s[i]` :
    - si c'est une lettre majuscule, la transformer avec ROT13
    - sinon le laisser tel quel
    - écrire ce caractère dans `t[i]`.
  - convertir le tableau `t` en chaîne avec `"".join(t)` (cf. l'opération `.join(...)` du cours, et afficher le résultat.

Pour tester qu'une lettre est majuscule ou minuscule, on peut comparer son code à ceux des caractères `'A'` `'Z'`. On rappelle que la fonction `ord()` permet d'obtenir le code d'un caractère.

On ne demande pas de transformer les lettres accentuées ni les minuscules.

- ◇ Transformer aussi les lettres minuscules.
- ◇ Pourquoi `ROT13(ROT13(s)) = s` ?

#### Réponse:

- Voir le fichier `exo3_1.py`
- Voir le fichier `exo3_2.py`
- Il y a 26 lettres dans l'alphabet. Si `x` est le code d'une lettre, alors :

$$\begin{aligned}
 x + 13 &\equiv k \pmod{26} && \text{(on applique ROT13 une première fois, on obtient un code } k) \\
 x + 13 + 13 &\equiv k + 13 \pmod{26} && \text{(on applique ROT13 une seconde fois)} \\
 x + 26 &\equiv k + 13 \pmod{26} \\
 x &\equiv k + 13 \pmod{26} && \text{(ajouter } n \text{ lorsque l'on calcule modulo } n \text{ revient à ne rien faire)}
 \end{aligned}$$

### Exercice 4

Télécharger le fichier `jungle2.txt` depuis la page du cours. L'enregistrer dans le répertoire TP5 à côté des fichiers Python de ce TP.

- Écrire un programme Python qui charge le fichier `jungle2.txt` et trouve la longueur du mot le plus long. Un mot est une suite de caractères, majuscule ou minuscule (sans accents ni caractères spéciaux). On ne considère pas les mots composés comme un seul mot. On ne demande pas de gérer les cas d'erreur.

---

1. C'est un cas particulier de chiffrement de Jules César, qui consiste à décaler toutes les lettres d'un message d'un nombre fixe de caractères.

2. ◇ En Python, la notation  $t[i:j]$  où  $t$  est un tableau ou une chaîne et  $i$  et  $j$  des entiers, renvoie le sous-tableau (ou la sous-chaîne) constitué des éléments aux indices  $i$  (inclus) à  $j$  (exclus). Utiliser cette notation pour que le programme précédent renvoie en plus le mot le plus long trouvé.

**Réponse:** Voir les fichiers `exo4_1.py` et `exo4_2.py`.

### Exercice 5 ◇

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`.