# N. S. I. Term. - DS02 (jeudi 20 octobre 2022)

## Exercice 1. Épreuves pratiques 2021 Sujet 5 – Exercice 1

On modélise la représentation binaire d'un entier non signé par un tableau d'entiers dont les éléments sont 0 ou 1. Par exemple, le tableau `[1, 0, 1, 0, 0, 1, 1]` représente l'écriture binaire de l'entier dont l'écriture décimale est

```
2**6 + 2**4 + 2**1 + 2**0 = 83.
```

À l'aide d'un parcours séquentiel, écrire la fonction `convertir` répondant aux spécifications suivantes :

```
def convertir(T):
    """
    T est un tableau d'entiers, dont les éléments sont 0 ou 1 et
    représentant un entier écrit en binaire. Renvoie l'écriture
    décimale de l'entier positif dont la représentation binaire
    est donnée par le tableau T
    """
```

Exemple :

```
>>> convertir([1, 0, 1, 0, 0, 1, 1])
83
>>> convertir([1, 0, 0, 0, 0, 0, 1, 0])
130
```

## Exercice 2. Épreuves pratiques 2021 Sujet 11 – Exercice 1

Écrire une fonction `conv_bin` qui prend en paramètre un entier positif `n` et renvoie un couple `(b,bit)` où :

- `b` est une liste d'entiers correspondant à la représentation binaire de `n` ;
- `bit` correspond aux nombre de bits qui constituent `b`.

Exemple :

```
>>> conv_bin(9)
([1,0,1,1],4)
```

Aide :

- l'opérateur `//` donne le quotient de la division euclidienne : `5//2` donne `2` ;
- l'opérateur `%` donne le reste de la division euclidienne : `5%2` donne `1` ;
- `append` est une méthode qui ajoute un élément à une liste existante :

  Soit `T=[5,2,4]`, alors `T.append(10)` ajoute 10 à la liste `T`. Ainsi, `T` devient `[5,2,4,10]`.

- `reverse` est une méthode qui renverse les éléments d'une liste.

  Soit `T=[5,2,4,10]`. Après `T.reverse()`, la liste devient `[10,4,2,5]`.

On remarquera qu'on récupère la représentation binaire d'un entier `n` en partant de la gauche en appliquant successivement les instructions :

```
b = n%2
n = n//2
```

répétées autant que nécessaire.

## Exercice 3.

On considère le dictionnaire suivant, représentant les données relatives à un élève :

```
eleve1 = {'Nom' : 'BERNARD', 'Prénom' : 'Claude', 'Genre' : 'M',
          'Date de naissance' : ''}
```

**Q1.** Écrire une instruction, utilisant la fonction `print`, pour afficher, en utilisant les données enregistrées dans le dictionnaire `eleve1`, le prénom et le nom de cet élève, sous la forme : `Claude BERNARD`.

**Q2.** Écrire une instruction, pour que la date de naissance de cet élève, non renseignée initialement, soit mise à la valeur 06/12/2007 (sous la forme d'une chaîne de caractères).

**Q3.** Écrire une instruction pour supprimer du dictionnaire toute information relative au genre de l'élève (clé et valeur).

**Q4.** Écrire une instruction pour que l'information sur la classe de l'élève soit ajoutée au dictionnaire. L'élève sera supposé être scolarisée dans la classe nommée T5.
**Q5.** À l'aide de quelle(s) instruction(s), grâce à un parcours par les couples (clé, valeur), peut-on afficher toutes les informations sur l'élève.

**Exercice 4.**

**Q1. [Préliminaire]** Par quelles instructions peut-on construire une liste `L0` de longueur 10 dont tous les éléments sont égaux à zéro ? Proposer trois méthodes, utilisant respectivement l'opération « `*` » pour les listes, la méthode `.append()`, une compréhension de liste.
**Q2.** Rappeler l'instruction pour créer un dictionnaire vide, nommé `d`.
**Q3.** Rappeler l'instruction pour ajouter à un dictionnaire vide, `d`, une clé `'a'` associée à la valeur 0.

On considère la liste `L = ['a', 'b', …, 'z']` constituée des 26 lettres de l'alphabet, en minuscule.
**Q4.** À l'aide d'un parcours de la liste `L`, et des instructions rappelées aux questions **Q2** et **Q3**, construire, le dictionnaire `docc = {'a' : 0, 'b' : 0, …, 'z' : 0}` en ajoutant les couples (clé, valeur) au fil du parcours de `L` (l'algorithme de construction de ce dictionnaire présente des analogies avec celui qui permet de construire, avec la méthode `.append()`, la liste demandée en question **Q1**).

On souhaite compter les nombres d'occurrences de chaque lettre de l'alphabet dans un texte, à l'aide d'un dictionnaire, initialement égal au dictionnaire construit en **Q4.**
Pour chacune des 26 lettres de l'alphabet, la valeur associée à la clé correspondant à la lettre représentera le nombre d'occurrences de la lettre dans le texte. Ainsi, par exemple, initialement, `docc['a']` vaudra 0, et à la fin de l'application de l'algorithme de comptage des occurrences, si `docc['a']` vaut 10, cela signifiera que la lettre « $a$ » apparaît 10 fois dans le texte.

On suppose que tous les caractères dans le texte sont, soit un espace, soit une des 26 lettres de l'alphabet, en minuscule.
**Q5.** On considère la fonction `compte_occ` dont le code (incomplet) est le suivant :

```
1. def compte_occ(texte):
2.     docc = {'a' : 0, 'b' : 0, …, 'z' : 0} ##ici le dictionnaire construit en Q4
3.     for ... : ## parcours de la chaîne « texte »
4.         if ... :
5.             docc[...] = ...
6.     return ...
```

Compléter le code de la fonction `compte_occ` pour qu'elle renvoie un dictionnaire donnant le nombre d'occurrence de chacune des 26 lettres de l'alphabet dans le texte.
On suppose que le dictionnaire `docc = {'a' : 0, 'b' : 0, …, 'z' : 0}` est construit et comporte bien 26 clés égales à chacune des 26 lettres de l'alphabet, associées à la valeur 0 (ne pas compléter la première ligne)

- Exemple de résultat d'un appel à la fonction :

```
>>> compte_occ('vive les vacances')
{'a' : 2, 'b' : 0, 'c' : 2, 'd' : 0, 'e' : 3, ..., 'z' : 0}
```

**Q6.** Si le nombre d'apparitions (occurrences) de la lettre `'a'` dans la chaîne `texte` est 10. Par quel calcul peut-on calculer la fréquence d'apparition de la lettre `'a'` dans la chaîne `texte` ?
**Q7.** Entre les lignes 5 et 6 du code de la fonction, ajouter des instructions pour que les valeurs dans le dictionnaire renvoyé, ne soient plus les nombres d'apparitions de chaque lettre dans le texte, mais la fréquence d'apparition de chaque lettre dans le texte.
On appliquera l'algorithme suivant :

- faire une copie du dictionnaire `docc` que l'on nommera `dfreq` ;
- à l'aide d'un parcours par les valeurs, déterminer le nombre de lettres dans la chaîne `texte` ;
- parcourir le dictionnaire `dfreq` pour changer les nombres d'apparitions en fréquences.

La fonction la fonction `compte_occ` pourra être renommée `calcule_freq` et renverra le dictionnaire `dfreq`.

**Exercice 5.**

L'objectif de cet exercice est de mettre en place une modélisation d'un jeu de labyrinthe en langage Python.

On décide de représenter un labyrinthe par un tableau carré de taille `n`, dans lequel les cases seront des 0 si l'on peut s'y déplacer et des 1 s'il s'agit d'un mur. Voici un exemple de représentation d'un labyrinthe :

```
laby=[[0,1,1,1,1,1,1,1,1,1],
      [0,0,0,0,0,0,0,1,0,1],
      [1,0,1,1,1,1,0,1,0,1],
      [1,0,1,0,0,0,0,0,0,1],
      [1,0,1,1,1,1,1,0,1,1],
      [1,0,1,0,0,0,1,0,1,1],
      [1,0,1,0,1,0,1,0,1,1],
      [1,0,1,1,1,0,1,0,1,1],
      [1,0,0,0,0,0,1,0,0,1],
      [1,1,1,1,1,1,1,1,0,0]]
```

L'entrée du labyrinthe se situe à la première case du tableau (celle en haut à gauche) et la sortie du labyrinthe se trouve à la dernière case (celle en bas à droite).

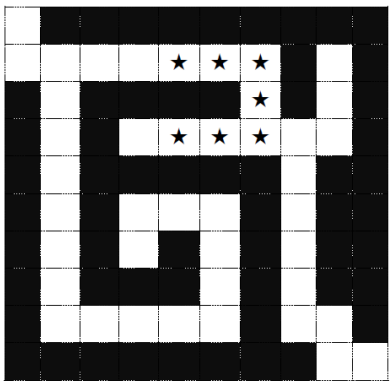
1. **Proposer**, en langage Python, une fonction `mur`, prenant en paramètre un tableau représentant un labyrinthe et deux entiers `i` et `j` compris entre 0 et `n-1` et qui renvoie un booléen indiquant la présence ou non d'un mur. Par exemple :

```
>>mur(laby, 2, 3)
True
>>mur(laby, 1, 8)
False
```

Un parcours dans le labyrinthe va être représenté par une liste de **cases.** Il s'agit de couples `(i,j)` où `i` et `j` correspondent respectivement aux numéros de ligne et de colonne des cases successivement visitées au long du parcours. Ainsi, la liste suivante

`[(1,4),(1,5),(1,6),(2,6),(3,6),(3,5),(3,4)]`

correspond au parcours repéré par des étoiles ⋆ ci-contre :



La liste `[(0,0),(1,0),(1,1),(5,1),(6,1)]` ne peut correspondre au parcours d'un labyrinthe car toutes les cases parcourues successivement ne sont pas adjacentes.

**2.** On considère la fonction `voisine` ci-dessous, écrite en langage Python, qui prend en paramètres deux cases données sous forme de couple.

```
def voisine(case1, case2) :
  l1, c1 = case1
  l2, c2 = case2
  # on vous rappelle que **2 signifie puissance 2
  d = (l1-l2)**2 + (c1-c2)**2
  return (d == 1)
```

2.a. Après avoir remarqué que les quantités `l1-l2` et `c1-c2` sont des entiers, **expliquer** pourquoi la fonction `voisine` indique si deux cases données sous forme de tuples `(l,c)` sont adjacentes.

2.b. **En déduire** une fonction `adjacentes` qui reçoit une liste de cases et renvoie un booléen indiquant si la liste des cases forme une chaîne de cases adjacentes.

Un parcours sera qualifié de ***compatible avec le labyrinthe*** lorsqu'il s'agit d'une succession de cases adjacentes accessibles (non murées). On donne la fonction `teste(cases, laby)` qui indique si le chemin `cases` est un chemin possible compatible avec le labyrinthe `laby` :

```
def teste(cases, laby) :
    if not adjacentes(cases) :
        return False
    possible = True
    i = 0
    while i < len(cases) and possible:
        if mur(laby, cases[i][0], cases[i][1]) :
            possible = False
        i = i + 1
    return possible
```

**3. Justifier** que la boucle de la fonction précédente se termine.

**4. En déduire** une fonction `echappe(cases, laby)` qui indique par un booléen si le chemin `cases` permet d'aller de l'entrée à la sortie du labyrinthe `laby`.