Séance A : Révisions

L1 – Université Côte d'Azur

Ceci est un sujet de révisions. Le but n'est pas tant d'approfondir vos connaissances mais de réviser les fondamentaux. Nous vous conseillons à chaque fois d'écrire votre réponse sur papier avant de vérifier la réponse avec Thonny.

Exercice 1 — Construire des type de bases (⋆)

Dans toutes les fonctions ci-dessous, on utilisera une boucle `for` et éventuellement les deux fonctions `ord` et `chr`.

1. Écrire une fonction `affiche_alphabet()` qui affiche les 26 lettres de l'alphabet.

```
def affiche_alphabet():
    for i in range(ord('a'),ord('z')+1):
        print(chr(i))
```

2. Écrire une fonction `créer_liste()` qui renvoie une liste contenant les 26 lettres de l'alphabet.

```
def créer_liste():
    L = [] # initialisation
    for i in range(ord('a'),ord('z')+1):
        lettre = chr(i)
        L.append(lettre) # ajout
    return L # renvoie
```

3. Écrire de même les fonctions `créer_tuple()`, `créer_chaîne()` et `créer_ensemble()` qui renvoient un tuple, une chaîne ou un ensemble contenant les 26 lettres de l'alphabet.

```
def créer_chaîne():
    C = "" # initialisation
    for i in range(ord('a'),ord('z')+1):
        lettre = chr(i)
        C = C + lettre # ajout
    return C # renvoie

def créer_tuple():
    T = () # initialisation
    for i in range(ord('a'),ord('z')+1):
        lettre = chr(i)
        T = T + (lettre,) # ajout (attention à la virgule)
    return T # renvoie

def créer_ensemble():
    E = set() # initialisation
    for i in range(ord('a'),ord('z')+1):
        lettre = chr(i)
        E.add(lettre) # ajout
    return E # renvoie
```

4. Écrire une fonction `créer_dico()` renvoyant un dictionnaire dont les clés sont les 26 lettres de l'alphabet minuscules et dont les valeurs sont les 26 lettres majuscules. On pourra utiliser la méthode `'a'.upper()`

```python
def créer_dico():
    D = dict() # initialisation
    for i in range(ord('a'),ord('z')+1):
        minuscule = chr(i)
        majuscule = minuscule.upper()
        D[minuscule] = majuscule # ajout
    return D # renvoie
```

Exercice 2 — Petites fonctions ultra-classiques (⋆)

1. Écrire les deux fonctions `mininum(a,b)` et `maximum(a,b)` qui prennent deux valeurs `a` et `b` et renvoient soit la plus petite, soit la plus grande.

```python
def minimum(a,b):
    if a<b:
        return a
    else:
        return b

def maximum(a,b):
    if a>b:
        return a
    else:
        return b
```

2. Même question, mais cette fois, les fonctions `min_liste(L)` et `max_liste(L)` prennent une liste en argument. Si la liste `L` est vide on renverra la valeur `None`.

```python
def min_liste(L):
    if L == []:
        return None
    else:
        m = L[0] # Existe car L est non vide !
        for i in range(len(L)):
            if L[i] < m:
                m = L[i]
        return m

def max_liste(L):
    if L == []:
        return None

    m = L[0] # Existe car L est non vide !
    for e in L:
        if e > m:
            m = e
    return m
```

3. Écrire les fonctions `somme(L)` et `produit(L)` qui renvoient la somme et le produit des éléments d'une liste.

```python
def somme(L):
    s = 0
    for i in range(len(L)):
        s = s + L[i]
    return s

def produit(L):
    p = 1 # Attention, pour le produit il faut initialiser à 1
    for e in L:
        p = p * e
    return p
```

4. Écrire une fonction `appartient(L,x)` qui renvoie `True` ou `False` suivant si `x` est un élément de `L` ou non.

```python
def appartient(L,x):
    for e in L:
        if x==e:
            return True
    return False
```

5. Écrire une fonction `indice(L,x)` qui renvoie l'indice de `x` dans `L` si `x` appartient à `L` et `None` sinon.

```python
def indice(L,x):
    for i in range(len(L)):
        if x==L[i]:
            return i
    return None
```

6. Écrire une fonction `positifs(L)` qui prend une liste d'entiers `L` en argument et qui renvoie `True` si tout les entiers de `L` sont positifs ou nuls ; sinon la fonction renverra `False`.

```python
def positifs(L):
    for i in range(len(L)):
        if L[i] < 0:
            return False
    return True
```

7. Écrire une fonction `double(L)` qui ne renvoie rien, mais modifie `L` en multipliant par 2 ses éléments.

```python
def double(L):
    for i in range(len(L)):
        L[i] = 2*L[i]
```

8. Écrire une fonction `fois_deux(L)` qui ne modifie pas `L` mais renvoie une nouvelle liste obtenue en multipliant les élements de `L` par 2.

```python
def fois_deux(L):
    L2 = []
    for i in range(len(L)):
        L2.append( 2*L[i] )
    return L2
```

Exercice 3 — Les boucles (⋆⋆)

On rappelle que pour obtenir un nombre aléatoire compris entre deux valeurs `a` et `b` en Python, il faut d'abord faire l'import avec `from random import randint` puis appeler `randint(a,b)`. Écrire une fonction `simulation()` qui simule une série de lancement d'un dé à 6 faces et qui s'arrête dès que la somme des valeurs obtenues est un multiple de 10 (non nul). Cette fonction devra afficher à chaque étape la nouvelle valeur aléatoire obtenue ainsi que la somme des valeurs. Elle renverra le nombre d'étapes qui aura été nécessaire afin d'obtenir un multiple de 10.

```python
from random import randint

def simulation():
    somme = 0
    nombre_d_étapes = 0
    while somme%10 != 0 or somme == 0:
        nombre_d_étapes = nombre_d_étapes + 1
        aléa = randint(1,6)
        somme = somme + aléa
        print(aléa,somme)
    return nombre_d_étapes
```

Exercice 4 — Construire des matrices (⋆⋆)

1. Écrire la fonction `matrice_nulle(n,m)` qui renvoie une matrice nulle de dimensions $n \times m$ ne contenant que des zéros et `dimensions(M)` renvoyant un couple `(n,m)` donnant le nombre de lignes et de colonnes de la matrice `M`.

```python
def matrice_nulle(i,j):
    M = []
    for ligne in range(i):
        L = []
        for colonne in range(j):
            L.append(0)
        M.append(L)
    return M

def dimensions(M):
    return (len(M),len(M[0]))
```

2. Écrire une fonction `afficher_matrice(M)` qui affiche proprement le contenu d'une matrice. On suppose que chaque valeur est comprise en 0 et 99 et tiens donc sur 2 caractères.

```
>>> M = [[1,2,3], [10,20,30], [11,22,33]]
>>> afficher_matrice(M)
 1  2  3
10 20 30
11 22 33
>>> f"{x: >2}" # Astuce (ici x=5) permet d'afficher x sur deux caractères
' 5'
```

```python
def afficher_matrice(M):
    (n,m) = dimensions(M)
    for i in range(n):
        for j in range(m):
            x=M[i][j]
            print(f"{x: >2}",end=' ')
        print()
```

3. Écrire des fonctions pour créer les matrices suivantes.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix} \quad \begin{pmatrix} 6 & 6 & 6 & 0 \\ 6 & 6 & 0 & 9 \\ 6 & 0 & 9 & 9 \\ 0 & 9 & 9 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18 & 20 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 & 27 & 30 \\ 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 & 36 & 40 \\ 5 & 10 & 15 & 20 & 25 & 30 & 35 & 40 & 45 & 50 \\ 6 & 12 & 18 & 24 & 30 & 36 & 42 & 48 & 54 & 60 \\ 7 & 14 & 21 & 28 & 35 & 42 & 49 & 56 & 63 & 70 \\ 8 & 16 & 24 & 32 & 40 & 48 & 56 & 64 & 72 & 80 \\ 9 & 18 & 27 & 36 & 45 & 54 & 63 & 72 & 81 & 90 \end{pmatrix}$$

```python
def mat1():
    (n,m) = (4,4)
    M = matrice_nulle(n,m)
    for i in range(n):
            M[i][i]=1
    return M

def mat2():
    (n,m) = (4,4)
    M = matrice_nulle(n,m)
    for i in range(n):
        for j in range(m):
            M[i][j]= i
    return M

def mat3():
    (n,m) = (4,4)
    M = matrice_nulle(n,m)
    for i in range(n):
        for j in range(m):
            M[i][j]= j+1
    return M

def mat4():
    (n,m) = (4,4)
    M = matrice_nulle(n,m)
    for i in range(n):
        for j in range(m):
            if i > m-j-1:
                M[i][j]= 9
            elif i < m-j-1:
                M[i][j]= 6
            else:
                M[i][j] = 0

    return M

def mat5():
    (n,m) = (9,10)
    M = matrice_nulle(n,m)
    for i in range(n):
        for j in range(m):
            M[i][j]= (i+1) * (j+1)
    return M

```

Exercice 5 — Petites fonctions ultra-classiques appliquées aux matrices (⋆)

Dans cette exercice on va reprendre les fonctions de l'exercice 2, mais en les adaptant aux matrices.

1. Écrire la fonction `minimum_matrice(M)` qui renvoie le plus petit élément d'une matrice.

```python
def minimum_matrice(M):
    (n,m) = dimensions(M)
    minimum = M[0][0] # Une matrice n'est jamais vide
    for i in range(n):
        for j in range(m):
            if M[i][j] < minimum:
                minimum = M[i][j]
    return minimum
```

2. Écrire une fonction `produit_matrice(M)` qui calcule le produit des éléments d'une matrice.

```python
def produit_matrice(M):
    (n,m) = dimensions(M)
    produit = 1
    for i in range(n):
        for j in range(m):
            produit = produit *  M[i][j]
    return produit
```

3. Écrire une fonction `appartient_matrice(M,x)` qui renvoie `True` ou `False` suivant si `x` est un élément de `M`.

```python
def appartient_matrice(M,x):
    (n,m) = dimensions(M)
    for i in range(n):
        for j in range(m):
            if M[i][j] == x:
                return True
    return False
```

4. Écrire une fonction `indice_matrice(M,x)` qui renvoie un couple `(i,j)` correspondant aux indices de `x` dans `M` si `x` appartient à `M` et `None` sinon.

```python
def indice_matrice(M,x):
    (n,m) = dimensions(M)
    for i in range(n):
        for j in range(m):
            if M[i][j] == x:
                return (i,j)
    return None
```

5. Écrire une fonction `positifs_matrice(M)` qui renvoie `True` si toutes les valeurs de M sont positives ou nulles.

```python
def positifs_matrice(M,x):
    (n,m) = dimensions(M)
    for i in range(n):
        for j in range(m):
            if M[i][j] < 0:
                return False
    return True
```

6. Écrire une fonction `fois_deux_matrice(M)` qui ne renvoie rien, mais modifie `M` en multipliant par 2 ses éléments.

```python
def fois_deux_matrice(M,x):
    (n,m) = dimensions(M)
    for i in range(n):
        for j in range(m):
            M[i][j] = 2 * M[i][j]
```

7. Écrire une fonction `double_matrice(M)` qui ne modifie pas `M` mais renvoie une nouvelle matrice obtenue en multipliant les élements de `M` par 2.

```python
def fois_deux_matrice(M,x):
    (n,m) = dimensions(M)
    D = matrice_nulle(n,m)
    for i in range(n):
        for j in range(m):
            D[i][j] = 2 * M[i][j]
    return D
```

Exercice 6 — Petites fonctions moins simples (⋆⋆)

1. On dit qu'un ensemble E de nombre est symétrique si pour tout entier $n \in E$ on a aussi $-n \in E$. Écrire une fonction `symétrie(ensemble)` qui renvoie `True` si `ensemble` est un ensemble d'entiers symétrique et `False` si c'est un ensemble d'entiers asymétrique. On lèvera une exception *ValueError: Un élément n'est pas entier* si l'ensemble contient autre chose que des entiers.

```python
def symétrie(ensemble):
    for n in E:
        if type(n) != type(1):
            raise ValueError("Un élément n'est pas entier")
        if not (-n in E):
            return False
    return True
```

2. Écrire une fonction `signe(T)` qui prend un tuple d'entiers `T` en argument et qui renvoie `True` si tout les entiers de `T` sont positifs ou nuls ; `False` si tous les entiers sont stictements négatifs ; `None` sinon (ou si le tuple est vide).

```python
# On ajoute une fonction auxilliaire
def même_signe(a,b):
    if a >= 0  and b >= 0:
        return True
    elif a < 0  and b < 0:
        return True
    else:
        return False

def signe(T):
    if T == ():
        return None

    for i  in range(len(T)): # On vérifie que tous les éléments ont le même signe
        if même_signe(T[0],T[i]) == False:
            return None
    return T[0] >= 0 # Renvoie True ou False selon le signe de T[0]
```

3. Écrire une fonction `comptage(chaîne)` qui prend une chaîne contenant des lettres et d'autres symboles et qui construit un dictionnaire qui donne le nombre d'occurences de chaque lettre.
   Par exemple, l'appel de `comptage("Choucroute !")` renverra le dictionnaire contenant comme clés les 26 lettres majuscules de l'alphabet.

```
>>> dico = comptage("Choucroute !") ; print(dico['C'], dico['H'])
2 1
>>> dico['!']
Traceback (most recent call last):
  File "<console>", line 1, in <module>
KeyError:  '!'
```

```python
def majuscule(c): # équivalent à c.upper()
    if ord('A') <= ord(c) and ord(c) <= ord('Z'): # c est une majuscule
        return c
    elif ord('a') <= ord(c) and ord(c) <= ord('z'): # c est une minuscule
        return chr(ord(c) - ord('a') + ord('A'))
    else: # c n'est pas une lettre
        return c

def comptage(chaîne):
    dico = dict()
    for i in range(ord('A'),ord('Z')+1):
        lettre = chr(i)
        dico[lettre] = 0

    for caractère in chaîne:
        c = majuscule(caractère)
        if c in dico: # si c est une clé (c'est-à-dire, ici,  une majucule)
            dico[c] = dico[c] +1

    return dico
```

Exercice 7 — Arbres (⋆)

Exercices 4, 5 et 6 du td 7.