TP 8 Python – Boucles imbriquées et chaînes de caractères

1. Boucles imbriquées

On a déjà vu un exemple de programme Python qui fait intervenir une boucle dans une boucle. Par exemple au TP6 :

```
liste = [[4, 5, 6], [1, 10, 14], [5, 19, 23]]
a = []
for i in liste:
    for j in i:
        a = a + [j]
print(a)
```

Prenons un exemple plus utile. On souhaite avoir un moyen simple de déterminer si un entier peut ou non s'écrire sous la forme d'une somme de deux carrés, c'est-à-dire $n = a^2 + b^2$ avec $a, b \in \mathbb{N}$.

Dans un premier temps, on se contente des entiers entre 1 et10. Pour commencer, nous allons donc construire une liste qui contient tous les nombres $i^2 + j^2$, avec $i, j \in [\![1, 10]\!]$. On peut ainsi proposer le code suivant :

```
# Code 1
sommescarrés = []
for i in range(11):
    for j in range(11):
        sommescarrés.append(i**2 + j**2)
```

Exercice 1. Pour valider votre compréhension de ce programme, prévoyez puis vérifiez les valeurs de :

`sommescarrés[0]` `sommescarrés[4]` `sommescarrés[11]` `sommescarrés[-1]`

Quelle est la longueur de la liste `sommescarrés` ? Vérifiez votre prédiction.

Pour éviter certaines redondances, on aurait pu par exemple se limiter à déterminer les sommes $i^2 + j^2$ pour $j \leq i$. On aboutit à une version améliorée du code précédent :

```
# Code 2
sommescarrés2 = []
for i in range(11):
    for j in range(i+1):
        sommescarrés2.append(i**2 + j**2)
```

Exercice 2. Pour valider votre compréhension de ce programme, prévoyez puis vérifiez les valeurs de :

`sommescarrés2[0]` `sommescarrés2[4]` `sommescarrés2[-1]`

Quelle est la longueur de la liste `sommescarrés2` ? Vérifiez votre prédiction.

Pour vérifier si un nombre quelconque `n` peut on non s'écrire comme la somme de deux carrés, on peut utiliser le procédé suivant :

- Pour toutes les valeurs de i et j pertinentes, on calcule $i^2 + j^2$ et on vérifie si on a $n = i^2 + j^2$.
- Si $n = i^2 + j^2$, alors c'était en fait vrai
- Si on est allé au bout sans trouver de i et j convenables, alors c'était faux !

On obtient par exemple la fonction suivante :

```python
# Code 3
def vérifie_somme_2_carrés(n):
    for i in range(n):
        for j in range(i+1):
            if n == i**2 + j**2:
                return True
    return False
```

Exercice 3. 1. À l'aide de la fonctions ci-dessus, vérifier si l'entier 9867 peut ou non s'écrire comme la somme de deux carrés.

2. Des entiers i, j convenables ne peuvent en réalité pas être plus grands que $\sqrt{n}$. Modifier la fonction à l'aide de cette nouvelle borne, afin de réaliser moins de calculs.
3. Comparer le temps d'exécution des deux versions de la fonction avec l'entier 9867.

Exercice 4. Calculer les sommes et produits suivantes à l'aide d'un programme Python :

a) $\displaystyle\sum_{k=0}^{10} k^k$ b) $\displaystyle\sum_{i=1}^{50}\sum_{j=1}^{50} ij$ c) $\displaystyle\sum_{i=1}^{20}\sum_{j=i}^{50} 2^{-(i+j)}$ d) $\displaystyle\prod_{0\leq k<l\leq 10} (i+j)$

Exercice 5. Tri à bulles. On souhaite trier une liste avec l'idée suivante : on parcourt la liste en échangeant deux termes consécutifs s'ils ne sont pas rangés dans le bon ordre. Une fois la liste parcourue entièrement, on reparcourt à nouveau la liste depuis le début, et on recommence l'opération autant de fois qu'il y a d'éléments dans la liste, car on peut montrer que, dans ce cas, la liste sera alors finalement triée.

Ainsi, on obtient sur l'exemple de la liste $[1, 10, 2, 7, 1]$:

1. Premier passage : $[1, 10, 2, 7, 1]$ est transformée en $[1, 2, 10, 7, 1]$, puis en $[1, 2, 7, 10, 1]$ puis en $[1, 2, 7, 1, 10]$;
2. Deuxième passage : $[1, 2, 7, 1, 10]$ est transformée en $[1, 2, 1, 7, 10]$
3. Troisième passage : $[1, 2, 1, 7, 10]$ est transformée en $[1, 1, 2, 7, 10]$
4. Quatrième et cinquième passages : $[1, 1, 2, 7, 10]$ n'est pas transformée.
   L'algorithme s'arrête (5 éléments $\rightarrow$ 5 itérations).

Coder en Python une fonction `tri_a_bulles(L)` qui prend en argument une liste `L` de nombres, et renvoie la liste `L` triée dans l'ordre croissant.

2. Chaînes de caractères

2.A) Cours

Définition

En Python, une **chaîne de caractères**, du type `str` ("string of characters") est une suite ordonnée de caractères typographiques, c'est-à-dire un bout de texte.
Une chaîne de caractères est définie par un texte donné entre guillemets ou entre apostrophes.

Remarque. ⚠ On préférera les guillemets (`"…"`), pour ne pas créer d'erreurs en mettant une apostrophe au milieu d'une chaîne de caractères.
Par exemple, l'instruction `maphrase = 'J'adore Python !'` renvoie une erreur, alors que `maphrase = "J'adore Python !"` est parfaitement valide.

Exemple

- `"J'adore Python !"` est une chaîne de caractères composée de 16 caractères.
- `maphrase = "Tous les c4r@ctères sont p&rmis~!"` stocke le texte indiqué, dans une variable nommée `maphrase`.

De la même manière qu'avec les listes, on peut :

- Accéder au caractère en position i de la chaîne `montexte`, avec l'instruction `montexte[i]`.
- Itérer sur les caractères d'une chaîne de caractère, avec l'instruction `for caractère in montexte:` (sans oublier d'indenter la suite !).
- Concaténer deux chaînes de caractères, avec l'opération `+`.
- Concaténer une chaîne avec elle même k fois, en utilisant `montexte * k`, ou `k * montexte`.
- Accéder à la longueur d'une chaîne de caractères, avec l'instruction `len(montexte)`
- Faire du *slicing* avec l'instruction `montexte[début:fin:pas]`
- ⚠ La méthode append n'existe pas pour les chaînes de caractères. Pour rajouter "a" à la fin de ma chaîne, je dois donc utiliser l'instruction `montexte = montexte + "a"`.
  Pour simplifier l'écriture, on écrira : `montexte += "a"`.

Remarque. La syntaxe `+=` fonctionne pour tous les ajouts en Python : ajouter un terme à une somme, ajouter deux listes, etc. On en reverra de nombreux exemples au cours de l'année.

Exemple

```
>>> maphrase = "Tous les c4r@ctères sont p&rmis~!"
>>> len(maphrase)
33
>>> maphrase[3]
's'
>>> maphrase[1:7]
'ous le'
>>> maphrase[4:15:2]
' e 4@t'
>>> mot1, mot2 = 'test', 'bonjour'
>>> mot1 + mot2
'testbonjour'
>>> mot2 + " " + mot1
'bonjour test'
>>> 4 * mot1
'testtesttesttest'
```

```
1 chaine = ""
2 for i in "bonjour":
3     chaine += i*3
4 print(chaine)
```

affichera `'bbbooonnnjjjooouuurrr'`

Dans une chaîne de caractères, un caractère a un rôle particulier, et est absolument à connaître : le caractère `"\n"`. Il indique un **retour à la ligne** dans le texte : le `n` signifie *newline* ("nouvelle ligne"). Attention, il est comptabilisé comme un **unique** caractère !

Exemple

```
>>> montexte = "Voici un texte.\nAvec un retour à la ligne !"
>>> print(montexte)
Voici un texte.
Avec un retour à la ligne !
>>> len(montexte)
43
>>> montexte[15]
'\n'
```

2.B) Exercices

Exercice 6. Écrire une fonction `liste_de_courses`, qui prend en argument une liste de chaînes de caractères, comme par exemple `["12 œufs", "farine", "PQ"]`, et qui l'affiche au format liste de course : chaque élément doit être sur une nouvelle ligne, précédé d'une espace et d'un tiret.
N'oubliez pas de tester votre fonction !

Exercice 7. Écrire une fonction `génère_mpd`, qui prend en argument un entier n, et qui renvoie un mot-de-passe aléatoire de n lettres. On utilisera :

- la chaîne de caractères `alphabet = "abcdefghijklmnopqrstuvwxyz"`
- la fonction `randint` du module `random`, importée avec : `from random import randint`

Dans la suite, vous pourrez tester vos fonctions avec des textes écrits par vos soins, mais il vous est aussi demandé de les tester avec deux textes que vous trouverez sur mon site, dans l'onglet informatique, et que vous copiez-collerez au début de votre fichier python.

Exercice 8.
1. Écrire une fonction `nombre_de_lignes(texte)`, qui prend en paramètre un texte sous forme d'une chaîne de caractères, et qui renvoie le nombre de lignes de ce texte.
2. Écrire une fonction `nombre_de_mots(texte)`, qui prend en paramètre un texte et qui renvoie le nombre de mots de ce texte. Attention, le texte peut contenir plusieurs lignes !

Exercice 9.
1. Écrire une fonction `recherche`, qui prend en argument une chaîne de caractères `chaine` et un caractère `c`, et qui renvoie :
   - Le booléen `False` si le caractère `c` n'est pas présent dans la chaîne ;
   - La position de la première apparition de `c` dans la chaîne sinon.
2. Écrire une fonction `positions`, qui prend en argument une chaîne de caractères `chaine` et un caractère `c`, et qui renvoie l'ensemble des positions du caractère `c` dans la chaîne `chaine`, sous la forme d'une liste.
3. (***) En déduire une fonction `découpe_mots`, qui prend en argument un texte contenant une seule ligne et qui renvoie la liste de ses mots. On considérera que les mots sont tous séparés par des espaces.

Exercice 10. Écrire une fonction `cherche_sous_chaine`, qui prend en argument deux chaînes de caractère `texte` et `mot`, et qui cherche si la sous-chaîne `mot` est présent dans la chaîne `texte`.
On utilisera des boucles `for` imbriquées.

Exercice 11. La fonction `str(obj)` permet de transformer un objet python en son équivalent sous la forme d'une chaîne de caractères. Par exemple, `str(128)` renvoie la chaîne `'128'` et `str(True)` renvoie `'True'`.
La fonction `int(ch)`, à l'inverse, permet de transformer une chaîne de caractères numériques en l'entier associé. Par exemple, `int('128')` renvoie l'entier 128.

1. Créer *par compréhension* (cf TP6) la liste des caractères correspondant aux chiffres de 0 à 9.
2. Compléter la fonction suivante, qui prend en argument un texte et renvoie la somme de tous les chiffres présents dans le texte.

```
def somme_chiffres(texte):
    somme = 0
    for c in texte:
        if ..............:
            somme += ............
    return somme
```

3. (****) Écrire une fonction qui prend en argument une chaîne de caractère du type `"123+45+79=250"` (avec uniquement des additions et un résultat), et qui renvoie un booléen qui indique si l'égalité est vraie ou fausse.
   Indication : On pourra utiliser et adapter les fonctions de l'exercice 9.

Exercice 12.

1. Écrire une fonction qui associe à chaque lettre l'entier entre 1 et 26 correspondant à sa position dans l'ordre alphabétique. On pourra utiliser la chaîne de caractères `alphabet` de l'exercice 7.
2. (*) Écrire une fonction `ordre_alphabétique(mot1, mot2)`, qui prend en argument deux mots (uniquement des lettres, sans accents) et qui renvoie un booléen qui indique si les deux mots sont ou non dans l'ordre alphabétique.
3. En déduire une fonction `est_triée_alpha(L)` qui prend en argument une **liste de mots** et qui renvoie un booléen qui indique si la liste est ou non triée par ordre alphabétique.
4. Comment pourrait-on trier une liste de mots dans l'ordre alphabétique ? Proposer un algorithme et le coder en Python.

Test des fonctions avec `lorem_ipsum` et `recette` :

Nom de la fonction	lorem_ipsum	recette
nombre_de_lignes	5	19
nombre_de_mots	120	363
recherche(..., "k")	False	11
positions(..., "k")	[ ]	[11, 1076, 1720, 1971, 2068]
positions(..., "x")	[477, 528, 585]	[189, 277, 309, 353, 415, 759, 840, 1457]
cherche_sous_chaine(..., "ips")	True	False
somme_chiffres	0	144