Séance 3 : Séquences et boucles for

L1 – Université Côte d'Azur

Dans ce TD, la seule boucle autorisée est le `for i in range(…)`. Sont interdit les boucles `while`, les boucles `for` sans `range` comme `for x in liste`, la méthode `append`, les indices négatifs et les compréhensions.

Exercice 1 — Jouons avec le `range`

```
for i in range(…) :
    print(i)
```

1. Dans le code ci-dessus, que faut-il ajouter à l'intérieur du `range(…)` dans le code ci-dessus de façon à afficher (avec un nombre par ligne et sans les virgules) :
   a) 1, 2, 3, 4, 5, 6, 7
   b) 1, 3, 5, 7, 9, 11, 13
   c) 17, 14, 11, 8, 5, 2, -1
2. Mêmes questions, mais avec le code ci-dessous en modifiant le `print`

```
for i in range(7) :
    print(…)
```

Exercice 2 — La disparition

1. Écrivez une fonction `nombre_apparitions(c,s)` qui prend en paramètre un caractère `c` et une chaîne de caractères `s` et qui renvoie le nombre de fois où `c` apparait dans `s`. Par exemple,

```
>>> nombre_apparitions('e','les revenentes')
5
```

2. En utilisant la fonction précédante, écrivez une fonction `absence_de_e(s)` qui prend en paramètre une chaîne de caractères `s` et renvoie `True` si `s` ne contient ni le caractère `e` ni `E`. Écrire des tests.
3. Si n est le nombre de caractères de `s`, quelle est la complexité [1] de votre solution ?
4. Proposez une autre solution qui n'a cette complexité que dans le cas où la fonction renvoie `True`, mais potentiellement une meilleure complexité quand elle renvoie `False`.

Exercice 3 — Un exercice renversant

1. Écrivez une fonction `affiche_miroir(s)` qui prend une chaîne de caractères `s` et qui **affiche** la chaîne `s` et son image miroir (`s` à l'envers) ; vous proposerez deux solutions, une avec un pas de boucle de -1, et l'autre avec un pas de boucle de 1.

```
>>> affiche_miroir('abc')
abc cba
```

2. Écrivez une fonction `miroir(s)` qui cette fois-ci **retourne** la chaîne de caractères `s` à l'envers, de sorte que l'on puisse répondre à la question 1 par

```
def affiche_miroir(s):
    print(s,miroir(s))
```

1. Pour évaluer la complexité, on évaluera le nombre d'accès mémoire : lecture/écriture de variable, lecture d'un caractère dans une chaîne de caractères, etc.

3. Écrivez une fonction `est_un_palindrome(s)` qui retourne `True` si `s` est un palindrome, autrement dit un mot qui est égal à son image miroir. Proposez une solution qui n'utilise pas la fonction `miroir` et qui ne crée aucune nouvelle chaîne de caractères. Écrire quatre tests avec `assert`.

Exercice 4 — Statistiques

On se donne une liste de notes `[12, 14.5, 9, 2, 19, "ABS",… ]`. On suppose que toutes les valeurs de la liste sont soit des nombres, soit la chaîne `"ABS"`.

1. Écrire une fonction `nombre_présents(liste_résultats)` qui renvoie le nombre de présents.

```
def nombre_présents(liste_résultats):
    n = 0
    for i in range(len(liste_résultats)):
        if liste_résultats[i] != "ABS":
            n = n+1
    return n
```

2. Écrire une fonction `moyenne(liste_résultats)` qui renvoie la moyenne des résultats. On considèrera qu'une absence équivaut à un zéro.

```
def moyenne(liste_résultats):
    somme = 0
    for i in range(len(liste_résultats)):
        if liste_résultats[i] != "ABS":
            somme = somme + liste_résultats[i]
    return somme/len(liste_résultats)
```

3. Écrire une fonction `bilan(liste_résultats)` qui renvoie une liste de trois nombres contenant : 1) la nombre de personne ayant validé l'UE, 2) le nombre de personnes ayant échouées tout en étant présentes 3) le nombre d'absent. On commencera par créer un tableau vide de trois cases que l'on modifiera à chaque tour de boucle.

```
def bilan(liste_résultats):
    tab = [0] * 3
    for i in range(len(liste_résultats)):
        if liste_résultats[i] == "ABS":
            tab[2] = tab[2]+1
        elif liste_résultats[i] <10:
            tab[1] = tab[1]+1
        else:
            tab[0] = tab[0]+1
    return tab
```

Exercice 5 — Entrelacement

Écrivez une fonction `entrelacement(s1,s2)` qui prend en paramètres deux chaînes de caractères `s1` et `s2` de même longueur et qui renvoie la chaîne qui contient en alternance un caractère de `s1` suivi d'un caractère de `s2`. Par exemple, `entrelacement('abc','123')` renvoie `'a1b2c3'`.

```
def entrelacement(s1,s2) :
    # Le assert est facultatif. Il permet de s'assurer que la condition est bien vérifiée.
    assert len(s1) == len(s2)
    res = ''
    for i in range(len(s1)) :
        res = res + s1[i] + s2[i]
    return res
```

Exercice 6 — Ponctuation

Écrivez une fonction `bien_ponctuée(s)` qui prend en paramètre une chaîne de caractères `s` et renvoie `True` si chaque point qui apparait dans la chaîne de caractères est suivi d'un espace, ou sinon est le dernier caractère de la chaîne.

```
>>> bien_ponctuée('Un. Deux.')
True
```

```
>>> bien_ponctuée('Trois.Quatre. ')
False
```

*Remarque : il est important de tester d'abord `i != len(s)-1` **avant** `s[i+1] != ' '`. Sinon, lorsque `i` sera égale à `len(s)-1`, l'évaluation de `s[i+1]` conduira à une erreur.*

```
def bien_ponctuée(s) :
    for i in range(len(s)) :
        if s[i] == '.' and i != len(s) - 1 and s[i+1] != ' ' :
            return False
    return True
```

une autre façon de l'écrire, qui n'utilise pas le connecteur logique `and`.

```
def bien_ponctuée_bis(s) :
    for i in range(len(s)) :
        if s[i] == '.' :
            if i != len(s) - 1 :
                if s[i+1] != ' ' :
                    return False
    return True
```

Exercice 7 — Pangrammes

Écrivez une fonction `est_pangramme(s)` qui prend en paramètre une chaîne de caractères `s` et qui renvoie `True` si `s` contient toutes les lettres de l'alphabet (on ne tient pas compte des caractères accentués).

```
>>> alphabet = 'abcdefghijklmnopqrstuvwxyz'
>>> est_pangramme(alphabet)
True
>>> est_pangramme('Portez ce vieux whisky au juge blond qui fume.')
True
```

Indication : vous pourrez utiliser la chaîne de caractères contenue dans la variable alphabet ci-dessus, ainsi que la méthode `s.lower()` pour mettre les lettres en minuscule.

```
def est_pangramme(s) :
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    s = s.lower()
    for i in range(len(alphabet)) :
        trouvé=False
        for j in range(len(s)):
            if s[j] == alphabet[i]:
                trouvé=True
                break
        if not trouvé:
            return False
    return True
```

La solution précédente est beaucoup trop complexe. Il est plus simple d'introdure une fonction auxiliaire :

```python
def est_présent(a,seq):
    for i in range(len(seq)):
        if seq[i] == a:
            return True
    return False

def est_pangramme_bis(s) :
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    s = s.lower()
    for i in range(len(alphabet)) :
        if not est_présent(alphabet[i], s):
            return False
    return True

```

Exercice 8 — Parenthèses

Un mot `w` est bien parenthésé si l'une des trois conditions suivantes est satisfaite :

- `w == ''`
- `w == '(' + w1 + ')'` et `w1` est bien parenthésé
- `w == w1 + w2` et `w1`, `w2` sont bien parenthésés

Écrivez une fonction `est_bien_parenthesée(s)` qui prend en argument une chaîne de caractères `s` contenant uniquement les caractères `'('` et `')'` et qui renvoie `True` si `s` est bien parenthésée.

```python
def est_bien_parenthesée(s) :
    n = 0 # <- n est le nombre de parenthèses ouvertes non encore fermées dans s[:i]
    for i in range(len(s)) :
        if s[i] == '(' :
            n = n + 1
        elif s[i] == ')' :
            n = n - 1
            if n < 0 :
                return False
    return n == 0
```