

Séance 3 : BOUCLES FOR ET CHAÎNES DE CARACTÈRES

L1 – Université Côte d’Azur

Boucle `for` ou boucle `while`?

De manière générale, il est préférable d’éviter d’utiliser une boucle `while` là où une boucle `for` suffit, à la fois par soucis de lisibilité et de **simplicité** du code (la boucle permet de s’assurer que l’on sortira de la boucle et évite de gérer la variable de boucle). Pour vous aider à prendre de bonnes habitudes, dans ce TD, il est **interdit d’utiliser la boucle `while`**.

Exercice 1 – Jouons avec le `range` (*)

1. On considère le code suivant :

```
1 for i in range(...) :
2     print(i)
```

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`

```
1 for i in range(7) :
2     print(...)
```

Exercice 2 – Comptine (*)

1. Écrivez une fonction `affiche_comptine(n)` qui prend en argument un entier n et qui **affiche** les n premières lignes d’une comptine un peu usante à la longue. On prendra garde à faire l’accord du pluriel.

```
1 >>> affiche_comptine(4)
2 "1 kilomètre à pied, c'est long."
3 "2 kilomètres à pied, c'est très long."
4 "3 kilomètres à pied, c'est très très long."
5 "4 kilomètres à pied, c'est très très très long."
```

2. Modifiez votre fonction pour en faire une fonction `comptine(n)` qui **renvoie** une unique chaîne de caractères contenant la comptine, de sorte que l’on puisse répondre à la première question en faisant simplement

```
1 def affiche_comptine(n) :
2     print(comptine(n))
```

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`.

```
1 >>> affiche_miroir('abc')
2 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

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

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 – 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` apparaît dans `s`. Par exemple,

```
1 >>> nombre_apparitions('e','les revenentes')
2 5
```

2. En déduire 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 complexité¹ 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 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'`.

Exercice 6 – Ponctuation (**)

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

```
1 >>> est_bien_ponctué('Un. Deux. ')
2 True
```

```
1 >>> est_bien_ponctué('Trois.Quatre. ')
2 False
```

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.

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).

```
1 >>> alphabet = 'abcdefghijklmnopqrstuvwxy'
2 >>> est_pangramme(alphabet)
3 True
4 >>> est_pangramme('Portez ce vieux whisky au juge blond qui fume.')
5 True
```

Indication : vous pourrez utiliser la chaîne de caractères contenue dans la variable `alphabet` ci-dessus, ainsi que la méthode `s.lower()`, ou (plus compliqué) vous générerez vous même la chaîne `alphabet` à l’aide de `chr` et `ord`.

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.