Séance 3: Boucles for et chaînes de caractères

L1 – Université Nice Sophia Antipolis

Objectifs:

- caractères de contrôle, échappement
- Mettre en oeuvre une boucle `for...in range`
- Parcourir une chaîne de caractères
- Parcourir plusieurs chaînes de caractères
- Générer une chaîne de caractères

For ou 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. Pour vous aider à prendre de bonnes habitudes, dans ce TD, il est **interdit d'utiliser la boucle while**.

Exercice 1 (Trop fort le range, ⋆)
On considère le code ci-dessous

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

```

Que faut-il ajouter à l'intérieur du `range(...)` *dans le code ci-dessus de façon à afficher*

1. *1, 2, 3, 4, 5, 6, 7 (chaque nombre sur une ligne différente, sans les virgules).*
2. *1, 3, 5, 7, 9, 11, 13*
3. *17, 14, 11, 8, 5, 2, -1*
4. *mêmes questions, mais avec le code à trou*

```
for i in range(7) :
    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.*

```
>>> affiche_comptine(4)
"1 kilomètre à pieds, c'est long."
"2 kilomètres à pieds, c'est très long."
"3 kilomètres à pieds, c'est très très long."
"4 kilomètres à pieds, 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* `def affiche_comptine(n) : print(comptine(n))`.

□

Exercice 3 (Un exercice renversant, ⋆)

1. *Écrivez une fonction* `show_mirror(s)` *qui prend une chaîne de caractères* `s` *et qui affiche* `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.

```
>>> show_mirror('abc')
abc cba
```

2. *Écrivez une fonction* `mirror(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 show_mirror(s): print(s,mirror(s))`.
3. *Écrivez une fonction* `is_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* `mirror` *et qui ne crée aucune nouvelle chaîne de caractères.*

□

Exercice 4 (La disparition, ⋆⋆)

1. *Écrivez une fonction* `nb_occurs(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,* `nb_occurs('e','les revenentes')` *renvoie 5.*
2. *En déduire une fonction* `has_no_e(s)` *qui prend en paramètre une chaîne de caractères* `s` *et renvoie* `True` *si* `s` *ne contient ni le caractère* `e` *ni* `E`.
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 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','def')` *renvoie* `'adbecf'`. □

Exercice 6 (Ponctuation, ⋆⋆)

Écrivez une fonction `est_bien_ponctue(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.*

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

```
>>> est_bien_ponctue('Trois.Quatre. ')
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).*

```
>>> 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(), ou (plus compliqué) vous chercherez à générer 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_parenthese(s)` *qui prend en argument une chaîne de caractères* `s` *contenant uniquement les caractères* `'('` *et* `')'` *et qui renvoie* `True` *si* `s` *est bien parenthésée.* □