Sequences : strings and lists

1 Going through a sequence

```python
def print_string(s):
    i = 0
    n = len(s)
    while i < n:
        print(s[i])
        i = i + 1

def print_string2(s):
    for i in range(0,len(s)):
        print(s[i], end='-')

def print_string3(s):
    for c in s:
        print(c, end='')
```

Classics on strings (also works with lists !)

Exercice 1 (Recherches)

1. Écrire une fonction qui compte le nombre d'occurrences d'un caractère dans une chaîne de caratères.
2. Écrire une fonction qui recherche si un caractère appartient à une chaîne. La fonction devra retourner la position du premier caractère trouvée, la valeur -1 si celui-ci n'est pas présent.

Exercice 2 (Palindrome)

Écrire une fonction qui détermine si une chaîne est un palindrome.

Quelques palindromes :

— Engage le jeu que je le gagne !

— Never odd or even.

— Nice hat, Bob Tahecin.

— God ! A red nugget ! A fat egg under a dog !

Deux niveaux :

level 1 : La chaîne contient uniquement des lettres minuscules non accentuées et des espaces. Le premier et le dernier caractères ne peuvent pas être des espaces, et il ne peut y avoir deux espaces qui se suivent.
Ex : "nice hat bob tahecin".

level + : La chaîne contient tout type de caractères : accentués, majuscules, ponctuation...
Ex : "Tu l'as trop écrasé' César, ce port salut."

Liste Itérative → Python

Type abstrait : Liste itérative	**Python : type list**
λ : Liste	`type(L) = list`
$\lambda \leftarrow$ *liste-vide*	`L = []`
longueur(λ)	`len(L)`
ième(λ, k)	`L[k]`

fonction `compte(Element x, Liste λ) : entier`
variables
    `entier  i, cpt`
debut
    `cpt ← 0`
    pour `i ← 1` **jusqu'a** `longueur(λ)` **faire**
        si `ieme(λ, i) = x` **alors**
            `cpt ← cpt + 1`
    fin pour
    retourne `cpt`
fin

```python
def count(x, L):
    cpt = 0
    for i in range(len(L)):
        if L[i] == x:
            cpt = cpt + 1
    return cpt
```

2 Lists are not strings

Exercice 3 (Construire une liste)

```
>>> help(list.append)
Help on method_descriptor:
append(...)
    L.append(object) -> None -- append object to end
>>> L = []
>>> L.append(1)
>>> L.append(2)
>>> L.append(3)
>>> L
[1, 2, 3]
```

Écrire une fonction qui retourne une nouvelle liste de n valeurs *val*.

Exercice 4 (Histogramme)

Voici quelques fonctions utiles pour cet exercice.

```
>>> help(ord)
    ord(c) -> integer
    Return the integer ordinal of a one-character string.

>>> ord('A')
65

>>> help(chr)
    chr(i) -> Unicode character
    Return a Unicode string of one character with ordinal i...

>>> chr(65)
'A'
```

1. Écrire une fonction qui donne un histogramme des caractères présents dans une chaîne de caractères : une liste de longueur 256 qui donne pour chaque caractère son nombre d'occurrences dans la chaîne.

2. Écrire une fonction qui compte le nombre de caractères différents dans une chaînes de caractères.
3. Écrire une fonction qui retourne le caractère le plus fréquent d'une chaîne, ainsi que son nombre d'occurrences.

---

3 Tris

Dans cette section, wikipédia est votre ami !

Exercice 5 (Tri par sélection (Select Sort))

Écrire la fonction `select_sort(`L`)` qui trie **en place** la liste L par la méthode du *tri par sélection.*

Questions + :

Quels problèmes pose l'implémentation **pas en place** du tri par sélection (en particulier en Python).

Exercice 6 (Tri par insertion (Insertion Sort))

1. Écrire une fonction qui insère un élément x à sa place dans une liste L triée.
2. Utiliser la fonction précédente pour écrire une fonction qui trie **pas en place** une liste.

Questions + :

— Comment implémenter **en place** le tri par insertion ?

— L'utilisation d'une recherche dichotomique améliore-t-elle vraiment le tri par insertion ?

bonus

Exercice 7 (Tri à bulles (Bubble Sort))

Implémenter en Python le *tri à bulles* (en place).

Questions + :

Modifier ce tri pour obtenir un *shaker sort*...

Exercice 8 (Tri fusion (Merge sort))

Pour trier une liste `L`, on procède (récursivement) de la façon suivante :

▷ Une liste de longueur < 2 est triée.

▷ Une liste de longueur ≥ 2 :

— on partitionne la liste `L` en deux sous-listes `L1` et `L2` de longueurs quasi identiques (à 1 près) ;

— on trie récursivement les deux listes `L1` et `L2` ;

— enfin, on fusionne les listes `L1` et `L2` en une liste triée.

Écrire la fonction `mergesort` qui trie en ordre croissant une liste (pas en place) avec la méthode ci-dessus.