1ère NSI — Exercices d'Entraînement

Parcours de listes en Python

Lycée Fustel de Coulanges, Massy

Marc Biver, décembre 2023, *v0.1*

Exercice 1: maximum d'une liste

Écrivez une fonction qui prend une liste de nombres positifs et retourne le plus grand nombre de cette liste.

Cet exercice est évidemment très proche de ce qu'on a fait en classe – sauf qu'au lieu de calculer une moyenne on va identifier un maximum.

Description simple de ce que notre fonction va faire :

- *Elle va initialiser une variable "Max" à 0 (parce que comme l'énoncé précise que les nombres sont positifs, on sait que leur maximum sera forcément supérieur ou égal à 0).*
- *Elle va parcourir la liste dans l'ordre de ses indices et, pour chacun d'entre eux elle va :*
  - *Si l'élément est strictement supérieur à Max, alors elle va remplacer Max par la valeur de l'élément.*
  - *Sinon elle va passer à l'élément suivant.*
- *Enfin, elle va retourner le Max ainsi identifié.*

Même chose, en pseudo-code un peu plus formalisé :

(manière plus structurée – donc valant plus de points – d'écrire la même chose)

Entrée: *liste dont tous les éléments* $\in \mathbb{R}+$
Sortie: $Max \in \mathbb{R}+$

```
1: fonction TrouveMax(liste)
2:     Max ← 0
3:     pour tout Element de liste faire
4:         si Element > Max alors
5:             Max ← Element
6:         fin si
7:     fin pour
8:     retourner Max
9: fin fonction
```

Et enfin en code Python :

```python
def TrouveMax(liste):
    # On initialise le maximum à 0
    Max = 0

    # On compte le nombre d'éléments de la liste
    NbElts = len(liste)

    # On parcourt la liste
    for i in range(NbElts):
        if liste[i] > Max:
            # On remplace par une nouvelle valeur de max
            Max = liste[i]

    # On a terminé,on retourne le max trouvé
    return Max
```

Exercice 2: minimum d'une liste

Écrivez une fonction qui prend une liste de nombres positifs et retourne le plus petit nombre de cette liste.

Cet exercice est évidemment quasiment identique au précédent – à une difficulté près : on ne sait pas comment initialiser notre variable Min (qui va jouer le même rôle que la variable Max dans l'exercice précédent). En effet, quelle que soit la valeur que l'on choisisse, on ne peut pas être sûr qu'elle soit supérieure au minimum réel.

Par exemple : si j'initie Min à 100 et que j'applique la même démarche que dans l'exercice précédent :

- *Sur la liste `[12, 200, 7, 89, 1000]`, cela fonctionnera puisque le minimum réel de la liste est inférieur à 100.*
- *En revanche sur la liste `[110, 1000, 200, 354]`, ça ne fonctionnera pas – la variable Min restera à 100, et la valeur retournée sera donc 100, ce qui est faux.*

On va donc devoir "ruser" en initialisant Min à la première valeur de la liste – puis poursuivre comme précédemment (sauf qu'on teste si l'élément est inférieur au min, au lieu de supérieur au max évidemment).

Pseudo-code :
Entrée: *liste dont tous les éléments* $\in \mathbb{R}+$
Sortie: $Min \in \mathbb{R}+$

```
fonction TrouveMax(liste)
    Max ← liste[0]
    pour tout Element de liste faire
        si Element < Min alors
            Min ← Element
        fin si
    fin pour
    retourner Min
```

9: ***fin fonction***

Code Python :

```
def TrouveMin(liste):
    # On initialise le minimum à la première valeur d'élément de la liste
    Min = liste[0]

    # On compte le nombre d'éléments de la liste
    NbElts = len(liste)

    # On parcourt la liste
    for i in range(NbElts):
        if liste[i] < Min:
            # On remplace par une nouvelle valeur de max
            Min = liste[i]

    # On a terminé,on retourne le max trouvé
    return Min
```

Exercice 3: vérification de la présence d'un élément dans une liste

Sans (évidemment) utiliser l'opérateur `in`, écrivez une fonction qui prend en entrée une liste de nombres et un nombre, et retourne `True` si l'élément est présent dans la liste , et `False` sinon.

Puisqu'on n'a pas le droit d'utiliser l'opérateur **`in`**, *il est évident qu'on va devoir parcourir la liste et vérifier un à un si ses éléments correspondent à celui qui a été passé en entrée.*

Pseudo-code :
Entrée: *liste dont tous les éléments* $\in \mathbb{R}$ *; et* $element_cherche \in \mathbb{R}$
Sortie: `Vrai` *ou* `Faux`

1: ***fonction*** EstPresent(*liste, element*)
2: resultat ← `Faux` ▷ *On part du principe que l'élément n'est pas présent, jusqu'à ce qu'on le trouve*
3: ***pour tout*** *Element de liste* ***faire***
4: ***si*** *Element* = $element_cherche$ ***alors***
5: resultat ← `Vrai`
6: ***fin si***
7: ***fin pour***
8: ***retourner*** *resultat*
9: ***fin fonction***

Remarque : ce code fonctionne et répond à l'énoncé mais il est très loin d'être optimisé – en effet, si on trouve l'élément dans le premier élément de la liste, on va continuer à parcourir tout le reste de la liste avant de renvoyer `Vrai`*, alors qu'on sait que c'est ce qu'on va faire depuis le début. C'est pour ça que je vous propose ci-dessous deux versions du code :*

— *La première est une application stricte du pseudo-code ci-dessus ; elle n'est*

pas optimale mais elle répond au besoin ;

— *La seconde utilise le fait que "return" force la sortie d'une fonction et est optimisée – mais pour les besoins du contrôle ce n'est pas grave si vous ne la comprenez pas.*

Code Python non optimisé :

```
def EstPresent(liste, element):
    # On initialise le résultat à Faux - on le changera si on trouve l'élément dans la liste
    resultat = False

    # On compte le nombre d'éléments de la liste
    NbElts = len(liste)

    # On parcourt la liste
    for i in range(NbElts):
        if liste[i] == element:
            # On a trouvé l'élément,le résultat va donc être vrai
            resultat = True

    # On a terminé,on retourne le max trouvé
    return resultat
```

Code Python optimisé :

```
def EstPresentOptim(liste, element):
    # On n'utilise pas de variable resultat cette fois - on va renvoyer directement la valeur depuis la boucle
    NbElts = len(liste)

    for i in range(NbElts):
        if liste[i] == element:
            # On a trouvé l'élément: on fait un "return" True directement. La commande "return" stoppe la fonction et donc la boucle - on ne va donc pas parcourir inutilement le reste de la liste.
            return True

    # Cette fois,si on est arrivé jusqu'ici,c'est qu'on n'a pas trouvé l'élément. On renvoie donc False
    return False
```

Exercice 4: liste de longueurs de mots

Écrivez une fonction qui prend une liste de mots et retourne une liste des longueurs correspondantes de ces mots.

Exercice qui devrait être assez trivial si vous avez bien assimilé les exercices

précédents.

Pseudo-code :

Entrée: *listeMots dont tous les éléments sont des chaînes de caractères (`str`)*
Sortie: *listeLongueurs dont les éléments sont les longueurs en nombre de caractères des chaînes de caractères présentes dans la liste passée en entrée*

1: ***fonction*** LONGUEURS*(listeMots)*
2: *listeLongueurs ← [liste vide]* ▷ *On initialise la liste qu'on renverra - pour le moment elle est vide*
3: ***pour tout*** *Element de listeMots* ***faire***
4: *Ajout à ListeLongeurs de la valeur longueur(Element)*
5: ***fin pour***
6: ***retourner*** *ListeLongueurs*
7: ***fin fonction***

Code Python :

```
def Longueurs(listeMots):
    # On initialise la liste de longueurs avec une liste vide
    listeLongueurs = []

    # Comme d'habitude,on compte les éléments de la liste
    NbElts = len(listeMots)

    # Puis on boucle sur tous les éléments
    for i in range(NbElts):
        # On ajoute la longueur du mot en cours avec la méthode append
        LongueurDuMot = len(listeMots[i])
        listeLongueurs.append(LongueurDuMot)

    # On renvoie la liste ainsi constituée
    return listeLongueurs
```