Informatique 1ere année, CPBX, TD machine 2: Boucle `for`, Listes

Carole Blanc, Paul Dorbec

1 Listes

Une liste est une suite d'objets entre crochets, séparés par des virgules. Par exemple, `[3,5,6,2,8,1]` est une liste d'entiers.
Voici quelques instructions disponible sur les listes :

`len(L)` : le nombre d'éléments de la liste

`L[i]` : le `i`ème élément de la liste (numérotés à partir de 0)

`[]` : une liste vide

`L.append(x)` : ajoute `x` à la fin de la liste `L`

`L=L+[x]` : ajoute `x` à la fin de la liste `L`

`L=[x]+L` : ajoute `x` au début de la liste `L`.

Question 1.1 *Dans la fenêtre de l'interpréteur, testez les instructions suivantes :*

```
u = [1,4,7,5,8]
print(u)
v = ["a","b","d"]
print(v)
w=['a', 5 , "toto", 2.4]
print(w)
u=u+[2]
print(u)
u=[22]+u
print(u)
```

Pour définir des listes, on peut utiliser l'instruction `range`.
La fonction `range` permet de générer des séquences[1] de nombres suivant une progression arithmétique. L'appel `range(a,b,c)` produit une séquence de nombre qui commence par `a`, s'arrête avant d'atteindre ou de dépasser `b`, et évolue de `c` en `c`. Par exemple, `range(2,14,3)` retourne `2,5,8,11` (sans atteindre 14). On peut utiliser moins de paramètres à l'appel de range, `range(a,b)` sous-entend `c=1` et `range(b)` sous-entend `a=0` et `c=1`.

Question 1.2 *Testez :*

```
print(list(range(1, 10)))
print(list(range(2, 20, 3)))
print(list(range(2, 21, 3)))
print(list(range(2, 22, 3)))
v=list(range(17, 3, -2))
print(v)
```

Retrouvez le fonctionnement de `range`*. Observez la subtilité du terme indiquant la fin du range (notamment entre 20, 21 et 22).*
En utilisant `range` *proposez une instruction qui produit la liste des entiers pairs inférieurs ou égaux à 100 en ordre croissant. Même question mais cette fois la liste doit être en ordre décroissant.*

Question 1.3 *En utilisant* `L[x]`*, on peut accéder à l'élément numéro* `x` *d'une liste* `L`*. Les n éléments d'une liste sont numérotés de 0 à* $n-1$*. Testez :*

```
u = [1,4,7,3,8]
u[3]=12
print(u)
```

Question 1.4 *Quel message d'erreur signifie qu'on cherche un élément qui n'existe pas dans la liste ?*

1. Ce ne sont pas tout à fait des listes (on peut produire une liste avec `list(range(a,b,c))`), mais la boucle `for` traite directement la sortie de `range` (et même plus efficacement !).

2 Boucles `for`

La boucle `for` permet de *parcourir* les éléments d'une liste ; en voici la syntaxe :

```
for variable in liste :
    corps de la boucle
```

Remarquez bien les deux points à la fin de la ligne et le fait que les instructions appartenant au corps de la boucle doivent être indentées (décalées vers la droite) par rapport au mot clef `for`.
L'évaluation par python d'une boucle `for` correspond à l'algorithme suivant :

1. calculer la valeur de la liste et la mémoriser ;
2. si tous les éléments de la liste ont été traités sortir de la boucle ;
3. sinon affecter la valeur du premier élément non traité à la variable de boucle ;
4. exécuter les instructions du corps de la boucle ;
5. recommencer le traitement à partir de l'étape 2.

Question 2.1 *Écrivez la fonction suivante et analysez ce qui est affiché :*

```
def test():
  for i in [2, 8, 5]:
    print(i, i * i)
```

Que fait l'instruction `for` *? Observez les messages d'erreur et l'emplacement présumé des erreurs quand vous oubliez les* `":"` *et quand vous indentez mal. La variable* `i` *est-elle définie après exécution de ces instructions ? Si oui, quelle est sa valeur ?*

Question 2.2 *Écrivez la fonction suivante et analysez ce qui est affiché :*

```
def test2():
  for x in ['a', 'b','c']:
    for i in [1,2,3,4]:
      print(x,i)
```

Observez le résultat. Que se passerait-il si on inversait l'ordre des `for` *(en corrigeant l'indentation) ?*

Question 2.3 *Modifiez la fonction précédente en utilisant un* `range` *pour générer les nombres de 1 à 4. (Notez qu'il n'est pas utile de placer les valeurs du range dans une liste.)*

Question 2.4 *On considère la fonction suivante :*

```
def mystere(n):
  s = 0
  for i in range(n+1):
     s = s + i
  return s
```

1. *Quelles sont les valeurs de* `mystere(3)`*, de* `mystere(5)` *et de* `mystere(n)` *?*
2. *Modifiez la fonction* `mystere` *en remplaçant l'instruction* `s = s + i` *par l'instruction* `s = s + 1` *dans la foucle* `for`*. Quelles sont les valeurs de* `mystere(3)`*, de* `mystere(5)` *et de* `mystere(n)` *après cette modification ?*

Notez que suite à cette modification la variable de boucle n'est plus utilisée dans le corps de la boucle mais que le même nombre d'itérations a été réalisé.

Question 2.5 *La fonction factorielle peut être définie de la manière suivante :*

$$\begin{cases} 0! & = 1, \\ n! & = 1 \times 2 \times \cdots \times n \text{ pour } n \geq 1. \end{cases}$$

Écrivez une fonction `factorielle(n)` *qui utilise cette définition pour calculer et retourner* $n!$*.*

Question 2.6 *Testez la fonction précédente en affichant les factorielles des nombres de 0 à 16, à l'aide d'une boucle* `for` *dans l'interpréteur.*

Question 2.7 *Écrivez une fonction* `sommeImpairs(k)` *qui calcule la somme des* k *premiers impairs (de 1 à* $2k-1$*). Qu'observez vous sur les valeurs retournées ?*

Question 2.8 *Écrivez une fonction* `sommeBizarre(n)` *qui calcule*

$$\sum_{i=1}^{n} (3i+1)^2$$

Question 2.9 *Écrivez une fonction* `sommePuissance2(k)` *qui calcule la somme des* k *premières puissance de 2 (de* 2^0 *à* 2^{k-1}*). Qu'observez vous sur les valeurs retournées ?*

3 Exploration de listes

Vous pouvez reprendre les fonctions demandées sur le sujet de Cours intégré 1 pour vous échauffer avant les questions suivantes.

Question 3.1 *Écrivez une fonction* `moyenne(L)` *qui calcule la moyenne des valeurs dans une liste d'entiers.*

Question 3.2 *Écrivez une fonction* `nbApparitions(x,L)` *qui retourne le nombre d'apparitions de la valeur de* `x` *dans la liste* `L`*.*

Question 3.3 *Écrivez une fonction* `doublons(L)` *qui renvoie* `True` *si l'un des éléments de la liste apparait au moins deux fois,* `False` *sinon.*

Question 3.4 *Écrivez une fonction* `maxDesImpairs(L)` *qui retourne le maximum des éléments impairs d'une liste* `L` *d'entiers positifs. Si la liste ne contient pas d'élément impair, on affichera "pas d'impairs".*

Question 3.5 *Écrivez une fonction* `minDesImpairs(L)` *qui retourne le minimum des éléments impairs d'une liste* `L` *d'entiers positifs. Si la liste ne contient pas d'élément impair, on affichera "pas d'impairs".*

Question 3.6 *Écrivez une fonction* `dernierPlusGrand(L)` *qui prend en entrée une liste* `L` *d'entiers et retourne la position dans* `L` *du dernier élément supérieur à la moyenne.*

Question 3.7 *Écrivez une fonction* `plusProche(L,n)` *qui prend en entrée une liste* `L` *d'entiers et retourne la valeur la plus proche de la moyenne parmi les éléments de la list, c'est à dire l'élément x qui minimise*

$$|\texttt{moyenne}(L) - x|$$

4 Pour les plus rapides (dur et facultatif)

Médiane : La médiane d'une liste est une valeur x de la liste telle que le nombre d'éléments de la liste de valeur supérieure ou égale à x est égal au nombre d'éléments inférieurs (-1 si la liste est de longueur impaire).

Question 4.1 *Écrivez une fonction qui retourne la médiane d'une liste. Quelle est la complexité de votre fonction ?*

Majorité absolue : Au premier tour des élections présidentielles, un candidat est élu s'il obtient plus de la moitié des voix (on dit alors qu'il a une majorité absolue).

Question 4.2 *Écrivez une fonction* `PremierTour(L)` *qui retourne l'élément de la liste qui apparaît dans plus de la moitié des cas s'il existe, et 0 si aucun élément n'est majoritaire.*

Question 4.3 *Combien de consultations d'un élément de la liste sont utilisées par votre algorithme ?*

On peut (probablement) améliorer l'algorithme que vous avez proposé avec la méthode suivante : On part avec une liste L et un stock S (vide au départ, et ne pouvant contenir que des éléments identiques). Pour chaque élément non traité de la liste, on procède itérativement ainsi :

- si le stock est vide, on place l'élément dans le stock.
- si le stock n'est pas vide et que l'élément est identique aux éléments du stock, on l'ajoute au stock.
- si le stock n'est pas vide mais que l'élément est différent des éléments du stock, on jette l'élément ainsi qu'un élément du stock.

Vous noterez que l'on peut modéliser le stock avec simplement deux variables : la valeur de l'élément et le nombre d'occurrences de cet élément dans le stock. On affirme que si un élément de la liste est majoritaire, c'est l'élément qui est dans le stock à la fin de l'algorithme. Néanmoins, l'élément du stock peut ne pas être majoritaire (si aucun élément ne l'est).

Question 4.4 *En vous appuyant sur ce qui précède, écrivez une nouvelle fonction* `PremierTour(L)` *qui applique l'algorithme précédent et teste ensuite si l'élément du stocke est majoritaire. Comparez les temps de calcul des deux fonctions sur de grandes listes.*