TP n° 03 – Manipulation des listes et recherche

I Manipulations de listes

Définition. Une liste est une structure de données qui contient une série de valeurs. Pour en déclarer une, on utilise la syntaxe suivante :

```
[1,2,3]
['a','x','mot']
[1,'chiffre',-2**3]
```

Exercice 1. Commandes de base.

1. Créer la liste `L=['a','b','c','d','e','f']`.
2. Afficher le résultat des commandes suivantes : `L[0]`, `L[1]`. Comment faire pour afficher l'élément `'d'` de `L` ?
3. Que renvoie `L[6]` ? Pourquoi ?
4. Que renvoie `L[2:5]` ? Que renvoie en général `L[i:j]` ? Et `L[i:]` ?
5. Que renvoie `len(L)` ? Comment faire pour afficher le dernier élément de `L` ?
6. Opérations sur les listes : `+` et `*`.
   Exécuter les commandes suivantes pour comprendre l'effet de ces opérations sur les listes (il faudra afficher le résultat des opérations pour constater leurs actions) :

   ```
   [1,2,3]+[7,8,9]
   [1,2]+[6,7,8,9]
   [0]*10
   ```

7. Méthodes sur les listes : `pop` et `append`.
   Exécuter les commandes suivantes pour comprendre l'effet de ces méthodes sur les listes :

   ```
   L=[1,2,3]
   L.pop()
   L.append(7)
   ```

Définition. Liste en compréhension
Une liste en compréhension est une liste dont le contenu est défini par filtrage du contenu d'une autre liste. Sa construction se rapproche de la notation ensembliste en mathématiques. Exemples :

en langage ensembliste :	$\mathcal{S}_1 = \{3n+2 \mid n \in \mathbb{N},\ n < 20\}$
en Python :	`S1=[3*n+2 for n in range(20)]`

Exercice 2.
Utilisez une liste en compréhension pour créer la liste des multiples de 5 entre 0 et 100.

Exercice 3. Soit `L1` la liste suivante : `L1=[12,-3,8,1.2,7]`.

1. Ajouter `10` à la fin de la liste. Afficher la nouvelle liste.
2. Afficher l'élément `1.2` de la liste.
3. Retirer l'élément `12` de la liste.
4. Modifier l'élément en position 1 de la liste par l'élément `77`.
   A ce stade, votre liste `L1` doit être : `[-3,77,1.2,7,10]`.
5. Créer la liste `L2=[-3,77,1.2,7,10,0,1,2,3,...,100]`.
6. Créer la liste des 30 premiers termes de `L2` sans ses 3 premiers termes.
7. Créer la liste `L2` suivie de 40 zéros.

Exercice 4. Utilisez une boucle `for` pour afficher un par un tous les termes de la liste `L=[2,8,-7,3]`. (il y a deux syntaxes possibles).

Exercice 5. Ecrire une fonction `renverser` qui à une liste renvoie la liste renversée.
On reprogramme ainsi la méthode `reverse` déjà implémentée dans Python.

```
>>> L=[2,8,-1,7]
>>> renverser(L)
[7,-1,8,2]
```

II La bibliothèque copy

Définition. Une bibliothèque est un ensemble de fonctions. Celles-ci sont regroupées et mises à disposition afin de pouvoir être utilisées sans avoir à les réécrire. Celles-ci permettent de faire : du calcul numérique, du graphisme,... Quelques exemples : les bibliothèques `math`, `matplotlib.pyplot`, `copy`.
Il existe plusieurs syntaxes pour importer une bibliothèque et pour ensuite dans le corps du programme, appeler les fonctions :

```
import math as m
m.cos(m.pi/3)
```

```
import math
math.cos(math.pi/3)
```

```
from math import cos, pi
cos(pi/3)
```

Exercice 6. Importez la bibliothèque `copy` sous l'alias `copy`.
Testez les fonctions suivantes. Expliquez les résultats :

```
L=[1,2,3,4]
M=L
L[0]=100
print(L,M)
print(id(L),id(M))
```

Programme n°1

```
L=[1,2,3,4]
M=copy.copy(L)
L[0]=100
print(L,M)
print(id(L),id(M))
```

Programme n°2

III Recherches

Exercice 7. Recherche dans une liste
Programmer une fonction `position(liste,element)` qui a comme entrée une liste et un élément et qui renvoie la position de l'élément dans la liste et qui renvoie `None` si l'élément n'est pas dans la liste.
On reprogramme ainsi la méthode `index` déjà implémentée dans Python

Exercice 8. Recherche du maximum dans une liste de nombres.

1. Ecrire une fonction `maximum(liste)` qui renvoie le maximum d'une liste de nombres non triée.

   ```
   >>>maximum([2,8,-7,3])
   8
   ```

2. Dans l'algorithme précédent, combien de fois parcourt-on la liste ?
3. Ecrire une fonction `maximum2(liste)` qui renvoie la deuxième valeur maximale d'une liste de nombres non triés, tous distincts.

   ```
   >>>maximum2([2,8,-7,3])
   3
   ```

Exercice 9. Pour les listes, il existe une fonction `max` et une méthode `index`. On en rappelle les syntaxes respectives dans l'annexe.
Testez `max` et `index` sur des exemples pour comprendre ce qu'elles renvoient.
A l'aide de `max` et `index`, déterminer la position du maximum de la liste `L`.

IV Exercices en plus

Exercice 10. Recherche d'un mot dans une chaîne de caractères.

1. Ecrire une fonction `estIci(motif,texte,i)` qui a comme entrée deux listes (ou deux chaînes de caractères) `motif` et `texte` et un entier `i` et qui renvoie `True` si `motif` est dans `texte` à la position `i` et `False` sinon.

   ```
   >>>estIci('le','Bonjour le monde',8)
   True
   >>>estIci('le','Bonjour le monde',9)
   False
   ```

2. Ecrire une fonction `recherche(motif,texte)` qui a comme entrée deux listes (ou deux chaînes de caractères) et qui renvoie `True` si `motif` est dans `texte` et `False` sinon.

```
>>>recherche('le','Bonjour le monde')
True
>>>recherche('bonjour','Bonjour le monde')
False
```

Exercice 11. Calendrier grégorien.
La manipulation des dates dans les logiciels de gestion, ou encore sur les sites web de réservation, doit s'effectuer conformément au calendrier grégorien (entré en vigueur à la fin du XVI^{e} siècle en France) en précisant pour chaque date le jour de la semaine. Or le calendrier grégorien est un format assez éloigné des formats habituels de stockage des nombres dans un ordinateur.
On cherche à déterminer, pour une date donnée, sa position dans l'énumération des jours depuis le 1er janvier 1600. et le jour de la semaine correspondant.

1. Proposer une fonction `nombre_de_jours(jours,mois,annee)` qui prend en entrée une date de la forme `jours,mois,annee` postérieure au 1,1,1600 et renvoie le nombre de jours écoulés entre le 1er janvier 1600 et la date considérée, sans tenir compte des années bissextiles (c'est-à-dire en supposant que chaque année comporte 365 jours). On pourra utiliser une liste des nombres de jours de chaque mois pour une année non bissextile : `m = [31,28,31,30,31,30,31,31,30,31,30,31]`.
2. Les années bissextiles sont déterminées par la règle suivante [1] :
   — Si l'année est divisible par 4 et non-divisible par 100, c'est une année bissextile (elle a 366 jours).
   — Si l'année est divisible par 400, c'est une année bissextile (elle a 366 jours).
   — Sinon, l'année n'est pas bissextile (elle a 365 jours).
   Proposer une fonction `bissextile(annee)` renvoyant `True` si l'année est bissextile et `False` sinon.
3. Modifier le programme de la fonction `nombre_de_jours(jours,mois,annee)` pour tenir compte des années bissextiles. Par exemple, `nombre_de_jours(1,2,1600)` doit retourner la valeur 31, `nombre_de_jours(1,1,1604)` la valeur 1461 (366+365+365+365) puisque l'année 1600 est bissextile.
4. Le 1er janvier 2001 était un lundi. Déterminer, en utilisant la fonction de la question précédente, quel jour de la semaine tombera le 1er mai 2040. Quel jour de la semaine est tombé le 14 juillet 1789 ?

1. Wikipédia en français, « Année bissextile », consulté le 23 novembre 2015.

Annexe

Fonctions et méthodes sur les listes

En Python, il y a deux façons de faire des opérations sur les objets qu'on manipule : les fonctions et les méthodes.
La différence est, pour vous, surtout d'ordre syntaxique.

Syntaxe fonction :	Syntaxe méthode :
`fonction(objet ,parametres)`	`objet.methode(parametres)`

Pour les listes, on trouve des fonctions et des méthodes. Certaines modifient la liste et ne renvoient rien, d'autres renvoient un résultat sans modifier la liste.

Exemples de méthodes pour les listes.
- **append(élément)** : modifie la liste en ajoutant élément à la fin de la liste
- **pop()** : modifie la liste en supprimant le dernier élément
- **reverse()** : modifie la liste en inversant les valeurs de la liste
- **index(élément)** : renvoie la position de élément dans la liste

Exemples de fonctions pour les listes :
- **len** : renvoie la longueur de la liste
- **max** : renvoie le maximum d'une liste de nombres

Fonctions et méthodes au programme

Opération	Exemple
Création d'une liste vide	`l=[]`
Création	`l=[1,56,13]`
Accès direct	`l[0]`
Extraction de tranche	`l[1:10]` `l[3:]`
Longueur	`len(l)`
Concaténation	`l1+l2`
Répétition	`[0]*k`
Modification par affectation	`l[0]=0`
Ajout en fin de liste	`l.append(1)`
Suppression en fin de liste	`l.pop()`
Création par compréhension	`[k**2 for k in range(n)]`
Copie	`copy(l)`

Fonctions et méthodes sur les chaînes de caractères

On retrouve certaines fonctions et méthodes pour les chaînes de caractères.

Opération	Exemple
Création d'une chaîne vide	`l=''`
Création	`s = 'Ma chaîne'`
Accès direct	`s[0]`
Extraction de tranche	`s[1:10]`
Longueur	`len(s)`
Concaténation	`s1+s2`
Répétition	`'e'*k`

Correction TP n° 03 – Manipulation des listes et recherche

Solution 1.
1. `L=['a','b','c','d','e','f']`.
2. `L[3]` renvoie `'d'`.
3. `L[6]` renvoie `out of range` : on a dépassé l'indice maximal de la liste.
4. `L[2:5]` renvoie `['c','d','e']`.
   `L[i:j]` est la sous-liste de `L` qui comprend les éléments d'indice i à $j-1$.
   `L[i:]` renvoie la sous-liste de `L` à partir de l'indice i.
5. `len(L)` renvoie la longueur de la liste. `L[len(L)]` renvoie le dernier terme.
6. `L1+L2` concatène les deux listes.
   `L1*n` répète n fois la liste `L`.
7. L.append(élément) : modifie la liste en ajoutant élément à la fin de la liste
8. pop() : modifie la liste en supprimant le dernier élément

Solution 2.
```
L=[5*n for n in range(21)]
```

Solution 3.
1. `L1.append(10)`
2. `print L1[3]`
3. `L1=L1[1:]`
4. `L1[1]=77`
5. `L2=L+[i for i in range(101)]`
6. `L3=L2[3:31]`
7. `L4=L2+40*[0]`

Solution 4.

```
for i in L:
    print(i)
```

ou

```
for i in range(len(L)):
        print(L[i])
```

Solution 5.

```
def renverser(liste):
    n=len(liste)
    listeRenversee=[]
    for i in range(n):
        listeRenversee.append(liste[n-i-1])
    return(listeRenversee)
```

Solution 6. Dans le programme n°1, il n'y a qu'un seul objet liste référencé par deux références : `L` et `M`. Si on modifie le contenu de `L`, la liste `M` est aussi modifiée. On le vérifie en affichant l'identité des objets référencés, qui est la même.
Dans le programme n°2, on crée un nouvel objet, avec une nouvelle référence.

Solution 7.

```
def f(L,a):
# on commence au début de la liste
    i=0
    # tant que on n'est pas au bout de la liste et que le terme de la liste n'est pas a, on avance
    while i<len(L) and L[i]!=a:
        i=i+1
    # si on est sorti de la liste, on renvoie None
```

```
    if i==len(L):
        return(None)
    # sinon on renvoie la position
    else:
        return(i)
```

Solution 8.

1.
```
def maximum(liste):
    n=len(liste)
    # le maximum est initialisé avec le premier terme de la liste
    max=liste[0]

    # on parcourt la liste
    for i in range(n):
    # si le i\` eme terme de la liste est supérieur au maximum, il devient la nouvelle valeur
        if liste[i]>max:
            max=liste[i]
    return(max)
```

2. Dans cette fonction, on compte le nombre d'opérations :
   — deux affectations avant la boucle
   — n itérations de la boucle dans laquelle on effectue :
      — un test
      — au pire des cas, une affectation.

   Au total, on a comme nombre d'opérations : $2 + 2n = O(n)$.

3.
```
def maximum2(L):
    Max=L[0]
    DeuxiemeMax=L[1]
    if Max<DeuxiemeMax:
        (Max,DeuxiemeMax)=(DeuxiemeMax,Max)
    for i in range(2,len(L)):
        if L[i]>Max:
            DeuxiemeMax=Max
            Max=L[i]
        elif:
            L[i]>DeuxiemeMax
            DeuxiemeMax=L[i]
    return(DeuxiemeMax)
```

Solution 9.
```
>>> L.index(max(L))
```

Solution 10. 1.

```
def estIci(motif,texte,i):
    k = 0
    p = len(motif)
    # on parcourt le texte \` a partir de l'index i tant que on ne dépasse pas la longueur de
    while k<p and motif[k] == texte[k+i]:
        k = k+1
    # si on a parcouru p termes, alors, motif est dans texte \` a l'index i
    if k==p:
        resultat=True
    else:
        resultat=False
    return resultat
```

2.

```
def recherche(motif,texte):
    n = len(texte)
    p = len(motif)
    # si motif est plus long que texte, on renvoie False
    if p>n:
        return False
    else:
    # par défaut, Resultat est False
        resultat=False
        i=0
        # on cherche motif dans texte \` a toutes les positions i
        while i <= n-p and resultat==False:
            resultat=estIci(motif,texte,i)
            i = i + 1
        return resultat
```

Solution 11. Calendrier grégorien.

1. Sans tenir compte des années bissextiles :
   — On compte 365 jours pour toutes les années entièrement écoulées.
   — On ajoute le nombre de jours des mois entièrement écoulés dans l'année en cours.
   — On ajoute le nombre de jours dans le mois en cours.

```
def nombre_de_jours_annees_classiques(jour,mois,annee):
  m = [31,28,31,30,31,30,31,31,30,31,30,31]
  nbjours = (annee-1600)*365 # nombre de jours ecoules pour les annees entieres
  for i in range(mois-1): # mois numerotes de 1 a 12 ; liste indexee de 0 a 11
    nbjours = nbjours + m[i]  # nombre de jours ecoules pour les mois termines
  nbjours = nbjours + jour - 1 # nombre de jours ecoules pour le mois en cours
  return nbjours
```

2. Détermination des années bissextiles.

```
def bissextile(annee):
  if annee % 4 == 0:
    if annee % 100 == 0:
      if annee % 400 == 0:
        bissex = True # annee multiple de 400
      else:
        bissex = False # annee multiple de 4 et de 100 mais pas de 400
    else:
      bissex = True # annee multiple de 4 mais pas de 100
  else:
    bissex = False # annee non multiple de 4
  return bissex
```

3. En tenant compte des années bissextiles :
   — On compte 365 ou 366 jours pour toutes les années entièrement écoulées selon qu'elles sont bissextiles ou non.
   — On ajoute le nombre de jours des mois classiques entièrement écoulés dans l'année en cours. On ajoute un jour si le mois de février de l'année en cours est terminé et que l'année en cours est bissextile.
   — On ajoute le nombre de jours dans le mois en cours.

```
def nombre_de_jours_toutes_annees(jour,mois,annee):
  m = [31,28,31,30,31,30,31,31,30,31,30,31]
  nbjours = 0
  for an in range(1600,annee): # pour tous les ans entierement ecoules
```

```
    if bissextile(an):
      nbjours = nbjours + 366 # si annee bissextile on ajoute 366
    else:
      nbjours = nbjours + 365 # sinon on ajoute 365
  for i in range(mois-1):
    nbjours = nbjours + m[i] # nombre de jours ecoules pour les mois termines
  if (mois > 3 or mois == 3) and bissextile(annee):
    nbjours = nbjours + 1 # on ajoute 1 si fevrier est termine et
                          # que l'annee en cours bissextile
  nbjours = nbjours + jour - 1 # nombre de jours ecoules pour le mois en cours
  return nbjours
```

4. Les semaines font toujours 7 jours. On détermine donc uniquement le nombre de jours écoulés depuis le 1er janvier 2001 modulo 7. Si la date testée est postérieure au 1er janvier 2001, le reste est négatif ce qui ne pose pas de souci à python qui gère les indices négatifs dans les listes (-1 étant l'indice du dernier élément, -2 celui de l'avant -dernier, etc.).

```
jours = ["lundi","mardi","mercredi","jeudi","vendredi","samedi","dimanche"]

print(jours[
( nombre_de_jours_toutes_annees(1,5,2040)
- nombre_de_jours_toutes_annees(1,1,2001) ) % 7
])

print(jours[
( nombre_de_jours_toutes_annees(14,7,1789)
- nombre_de_jours_toutes_annees(1,1,2001) ) % 7
])
```