Listes Python

INFORMATIQUE COMMUNE - TP n° 1.3 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ créer une liste in extenso, avec une boucle ou en compréhension
- ☞ manipuler une liste pour ajouter, ôter ou sélectionner des éléments
- ☞ tester l'appartenance d'un élément à une liste
- ☞ utiliser la concaténation et le tronçonnage sur une liste
- ☞ itérer sur les éléments d'une liste avec une boucle for
- ☞ coder les algorithmes incontournables (count, max, min, sum, avg)

Les listes Python constituent le couteau suisse du langage et permettent de modéliser une collection d'informations. Selon l'épreuve que vous passerez au concours, elles représenteront des points GPS d'une trajectoire, l'orientation de spins dans un matériau ferromagnétique, la population d'un pays, la durée de vie d'un isotope, les nombres d'une conjecture, un jeu de dominos, un plateau de jeu de dames ou de solitaire...

A Jouons avec les listes

Toute utilisation d'une structure de données commence par une initialisation. C'est l'objet de cette section : initialiser correctement une liste [1].

A1. **Création et manipulation :**

(a) Créer **in extenso** [2] une liste contenant les 15 premiers entiers.

(b) Créer une liste contenant les 15 premiers entiers en utilisant une boucle `for`.

(c) Créer en compréhension une liste contenant les 15 premiers entiers.

(d) Sélectionner et afficher le cinquième et le neuvième élément.

(e) Tester l'appartenance des nombres 5 et 42 à la liste.

(f) Ajouter un élément à l'aide de la méthode append.

(g) Retirer le dernier élément.

(h) Afficher un élément sur deux à partir du deuxième.

(i) Concaténer la liste à elle même.

(j) Démultiplier la liste par cinq et afficher la longueur de la liste.

1. C'est aussi souvent le début des épreuves de concours...
2. C'est à dire en explicitant tous les éléments.

Solution :

Code 1 – Création et manipulation de listes

```
# In extenso
L = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
print(L)

# for loop
L = []
for i in range(15):
    L.append(i)
print(L)

# comprehension list
L = [i for i in range(15)]
print(L)

print(L[4], L[8])  # indexing

if 5 in L:
    print("5 is in list")

if 42 not in L:
    print("42 is not in list")

L.append(42)  # append to the list
print(L)

L.pop()  # remove last element
print(L)

# slicing - un sur deux à partir du premier
M = L[1::2]
print(M)

# slicing - un sur deux à partir du premier, en exluant les dernier
M = L[1:-1:2]
print(M)


L += L  # concatenation
print(L)

L *= 5  # demultiplication
print(len(L))
```

On suppose dans ce qui suit que n et h sont des variables telles que `h = 5` et `n = h * h`.

A2. Créer la liste des n premiers éléments pairs.

A3. Créer la liste des n premiers éléments impairs.

A4. Créer une liste de n éléments selon le modèle $[1, 4, 7, 10, \ldots]$.

A5. Créer une liste de n éléments ne contenant que des zéros de deux manières différentes.

A6. Créer une liste de n éléments ne contenant alternativement que 1 et -1 de deux manières différentes. C'est à dire : [1, -1, 1, -1 ...].

A7. Créer une liste de `n = h * h` éléments constituée de l'alternance de h fois 1 et de h fois -1 de deux manières différentes. La liste commence par un 1. [1, 1, 1,...,-1, -1, -1,..., 1, 1, 1, ...]

A8. Créer une liste de liste de `h` éléments constituée de l'alternance d'une liste de h fois 1 et d'une liste de h fois -1 de deux manières différentes. [[1, 1, 1, ... 1], [-1, -1, -1, ...,-1], [1, 1, 1, ..., 1], ...]

A9. Créer un jeu de dominos. On représente un domino par un tuple (2,3), le zéro marque l'absence de numéro. Le jeu contenant toutes les pièces est une liste de tuples. Le jeu de dominos ne contient que 28 éléments : `[(0, 0), (1, 0), (1, 1), (2, 0), (2, 1), (2, 2), (3, 0), (3, 1), (3, 2), (3, 3), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)]`

Solution :

Code 2 – Listes in extenso ou en comprehension

```
h = 5
n = h * h

# EVENS
L = []
for i in range(n):
    L.append(2 * i)  # in extenso
print(L)
L = [2 * i for i in range(n)]  # comprehension
print(L)

# ODDS
L = []
for i in range(n):
    L.append(2 * i + 1)  # in extenso
print(L)
L = [2 * i + 1 for i in range(n)]  # comprehension
print(L)

# [1, 4, 7, 10, ...]
L = []
start = 1
for i in range(n):
    L.append(start + 3 * i)  # in extenso
print(L)
L = [start + 3 * i for i in range(n)]  # comprehension
print(L)

# ZEROS
L = []
for i in range(n):
    L.append(0)  # in extenso
print(L)
L = [0] * n  # comprehension
print(L)

```

```
# [1, -1, 1, -1, ...]
# in extenso
L = []
for i in range(n):
    L.append((-1) ** i)  # in extenso
print(L)

# [1, -1, 1, -1, ...]
# comprehension
L = [(-1) ** i for i in range(n)]  # comprehension
print(L)

# [1, 1, 1, ... 1, -1, -1, -1, ..., 1, 1, 1, ...]
# in extenso
L = []
for i in range(h):
    for k in range(h):
        if i % 2 == 0:
            L.append(1)
        else:
            L.append(-1)
print(L)

# [1, 1, 1, ... 1, -1, -1, -1, ..., 1, 1, 1, ...]
# idem
# comprehension et concaténation
L = []
pu = [1] * h
mu = [-1] * h
for i in range(h):
    if i % 2 == 0:
        L += pu
    else:
        L += mu
print(L)

# [1, 1, 1, ... 1, -1, -1, -1, ..., 1, 1, 1, ...]
# idem
# comprehension uniquement
L = [(-1) ** i for i in range(h) for j in range(h)]  # comprehension
print(L)

# [ [1, 1, 1, ... 1], [-1, -1, -1, ...,-1], [1, 1, 1, ..., 1], ...]
# in extenso
L = []
for i in range(h):
    M = []
    for k in range(h):
        if i % 2 == 0:
            M.append(1)
        else:
            M.append(-1)  # in extenso
    L.append(M)
print(L)

```

```python
# [ [1, 1, 1, ... 1], [-1, -1, -1, ...,-1], [1, 1, 1, ..., 1], ...]
# idem
# partie en compréhension
L = []
pu = [1] * h  # comprehension
mu = [-1] * h  # comprehension
for i in range(h):
    if i % 2 == 0:
        L.append(pu)
    else:
        L.append(mu)
print(L)

# [ [1, 1, 1, ... 1], [-1, -1, -1, ...,-1], [1, 1, 1, ..., 1], ...]
# idem
# compréhension
L = [[(-1) ** i] * h for i in range(h)]  # comprehension
print(L)

# DOMINOS
# in extenso
dominos = []
for i in range(7):
    for j in range(i + 1):
        dominos.append((i, j))
print(len(dominos), dominos)

# DOMINOS
# comprehension
dominos = [(i, j) for i in range(7) for j in range(i + 1)]
print(len(dominos), dominos)
```

B Algorithmes incontournables

B1. Créer une liste L de dix nombres aléatoirement choisis dans l'intervalle $[0,1[$.

B2. Créer une fonction qui renvoie le nombre d'éléments strictement supérieurs à la valeur `v` dans une liste. Le prototype de cette fonction est `count_if_sup(L, v)`, où L est un type `list` et `v` un type `float`. Elle renvoie un type `int`.

B3. Créer une fonction qui renvoie l'élément maximal d'une liste s'il existe (liste non vide) et `None` sinon. Le prototype de cette fonction est `max_val(L)`, où L est un type `list`.

B4. Créer une fonction qui renvoie l'indice de l'élément maximal d'une liste s'il existe (liste non vide) et `None` sinon. Le prototype de cette fonction est `max_index(L)`, où L est un type `list`.

B5. Créer une fonction qui renvoie les indices des éléments minimal et maximal d'une liste sous la forme d'un tuple. Le prototype de cette fonction est `min_max_index(L)`, où L est un type `list`.

B6. Créer une fonction qui renvoie la moyenne des éléments d'une liste. Le prototype de cette fonction est `average(L)`, où L est un type `list`. Elle renvoie un type `float`.

Solution :

Code 3 – Algorithmes incontournables

```
import random


def count_if_sup(L, v):
    c = 0
    for elem in L:
        if elem > v:
            c += 1
    return c


def max_val(L):
    if len(L) > 0:
        maxi = L[0]
        for elem in L:
            if elem > maxi:
                maxi = elem
        return maxi
    else:
        return None


def max_index(L):
    if len(L) > 0:
        maxi = L[0]
        index = 0
        for i in range(1, len(L)):
            if L[i] > maxi:
                maxi = L[i]
                index = i
        return index
    else:
        return None


def min_max_index(L):
    if len(L) > 0:
        mini, maxi = L[0], L[0]
        i_min, i_max = 0, 0
        for i in range(1, len(L)):
            if L[i] > maxi:
                maxi = L[i]
                i_max = i
            if L[i] < mini:
                mini = L[i]
                i_min = i
        return (i_min, i_max)
    else:
        return None, None


def average(L):
```

```
    if len(L) > 0:
        acc = 0
        for elem in L:
            acc +=elem
        return acc/len(L)
    else:
        return None



# MAIN PROGRAM
N = 10
# in extenso
L = []
for i in range(N):
    L.append(random.random())
# idem comprehension
L = [random.random() for i in range(N)]

print(L)
threshold = 0.5
print(count_if_sup(L, threshold), " elements are greater than ", threshold)
print("Max value is ", max_val(L))
print("Max index is ", max_index(L))
print("(i_min, i_max) =", min_max_index(L))
i_min, i_max = min_max_index(L)  # unpacking tuple
print("i min =", i_min, ", i_max =", i_max)
print(average(L))
```

C Syracuse

On considère la suite u définie par $u_0 \in \mathbb{N}^\star$ et :

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \begin{cases} 3u_n + 1 & \text{si } u_n \text{ est impair} \\ \frac{u_n}{2} & \text{si } u_n \text{ est pair} \end{cases} \tag{1}$$

Compléter à la main :

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13
u_n	3													

La conjecture de Syracuse fait l'hypothèse qu'il existe un rang N tel que $u_N = 1$, quel que soit l'entier u_0 choisi. Pour un entier u_0 donné, on nomme :

durée du vol l'entier défini par : $\min\{k \in \mathbb{N}^\star, u_k = 1\}$

altitude maximale du vol l'entier défini par : $\max\{u_k, k \in \mathbb{N}\}$

durée de vol en altitude l'entier défini par : $\min\{k \in \mathbb{N}, u_{k+1} < u_0\}$

C1. Que valent la durée du vol, l'altitude maximale et la durée de vol en altitude pour $u_0 = 3$?

Solution : La durée du vol vaut 7, l'altitude maximale 16 et la durée du vol en altitude 5.

C2. Écrire une fonction `suivant(u)` prenant en entrée un type `int` et renvoyant le terme suivant. Par exemple `suivant(8)` renverra 4 et `suivant(3)` renverra 10.

C3. En utilisant la fonction précédente, écrire une fonction `syracuse(u0)` prenant comme paramètres d'entrée le premier terme `u0` de la suite de type `int`. Cette fonction renvoie la liste `vol` contenant de tous les termes u_i jusqu'à la durée du vol N, c'est à dire $u_N = 1$.

C4. Comment peut-on calculer la durée du vol à partir de la fonction précédente ?

Solution : La durée de vol est la quantité `len(vol)-1`.

C5. Modifier la fonction `syracuse(u0)` afin qu'elle renvoie également la durée de vol en altitude et l'altitude maximale. La valeur renvoyée sera un tuple `(vol, vol_alt, max_alt)`.

C6. Compléter :

u_0	7	25	54	97	703
Durée du vol					
Altitude maximale					
Durée de vol en altitude					

Solution :

Code 4 – J'aimerais tant voir Syracuse

```
def suivant(u):
    if u % 2 == 0:
        return u // 2
    else:
        return 3 * u + 1


def syracuse(u0):
    u = u0
    vol = [u0]
    while u != 1:
        u = suivant(u)
        vol.append(u)
    return vol


def steroids_syracuse(u0):
    u = u0
    vol = [u0]
    vol_alt = 0
    max_alt = u0
    flying = True
    while u != 1:
        u = suivant(u)
```

```
        vol.append(u)
        if u > max_alt:
            max_alt = u
        if flying:
            if u > u0:
                vol_alt += 1
            else:
                flying = False
    return vol, max_alt, vol_alt


if __name__ == "__main__":
    vol = syracuse(3)
    print(vol, len(vol)-1)

    for u0 in [3, 7, 25, 54, 97, 703]:
        vol, max_alt, vol_alt = steroids_syracuse(u0)
        print("u0 =", u0, "vol :", vol, " flight time :", len(vol)-1, " Max
            altitude :", max_alt, " alt. vol :", vol_alt)
```

D Comptage par dictionnaire

Il existe une structure de donnée utile pour rapidement compter les éléments d'une liste : le dictionnaire.

■ **Définition 1 — Dictionnaire.** Un dictionnaire est une structure de données dont les éléments $\mathcal{V}$, au lieu d'être indicés par un entier sont indicés par des clefs appartenant à un ensemble $\mathcal{K}$. Soit $k \in \mathcal{K}$, une clef d'un dictionnaire `D`. Alors `D[k]` est la valeur v de $\mathcal{V}$ qui correspond à la clef k.

On dit qu'un dictionnaire est un **tableau associatif** qui associe une clef `k` à une valeur `v`.

Les opérations sur un dictionnaire sont :

1. rechercher la présence d'une clef dans le dictionnaire,
2. accéder à la valeur correspondant à une clef,
3. insérer une valeur associée à une clef dans le dictionnaire,
4. supprimer une valeur associée à une clef dans le dictionnaire.

P Un dictionnaire relie donc directement une clef, qui n'est pas nécessairement un entier, à une valeur : pas besoin d'index intermédiaire pour rechercher une valeur comme dans une liste ou un tableau. Par contre, cette clef est nécessairement d'un type immuable.

Considérons l'exemple donnée sur l'exemple de la figure 1. On suppose que les éléments chimiques sont enregistrés via une chaîne de caractères : `"C"`, `"O"`,`"H"`, `"Cl"`, `"Ar"`, `"N"`. Soit `d`, un dictionnaire correspondant à la figure 1. Accéder au numéro atomique de l'élément `C` s'écrit : d["C"].

R Un dictionnaire n'est pas une structure ordonnée, à la différence des listes ou des tableaux.

À tout seigneur tout honneur, les accolades sont la voie royale pour créer un dictionnaire.

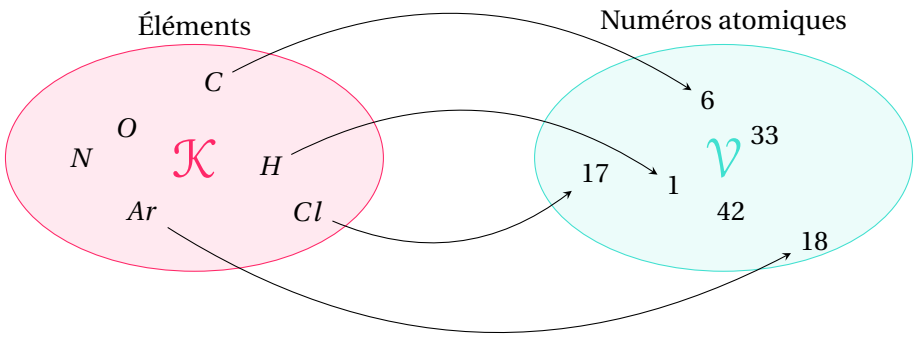

FIGURE 1 – Illustration du concept de dictionnaire, ensembles concrets

```
d = {}  # Empty dict
print(d, type(d))  # {} <class 'dict'>
d = {"Ar": 18, "Na": 11, "Cl": 17, "H": 1}
# keys are strings, values integers
print(d)  # {'Ar': 18, 'Na': 11, 'Cl': 17, 'H': 1}
```

La fonction `dict()` permet également de créer un dictionnaire à partir de n'importe quel objet itérable. Elle s'utilise le plus souvent pour convertir un liste de tuples en dictionnaire.

```
d = dict()  # Empty dict
print(d, type(d))  # {} <class 'dict'>
d = dict([("Ar", 18), ("Na", 11), ("Cl", 17), ("H", 1)])
# keys are strings, values integers
print(d)  # {'Ar': 18, 'Na': 11, 'Cl': 17, 'H': 1}
```

On peut connaître la taille d'un dictionnaire à l'aide de la fonction `len` et savoir si un élément est une clef d'un dictionnaire en utilisant l'opérateur `IN`.

```
d = {"Ar": 18, "Na": 11, "Cl": 17, "H": 1}
if  "Ar" in d:
  print("Ar", d["Ar"]) # Ar 18
if "O" not in d:
  print("O is not in d")  # O is not in d
```

D1. Construire une liste de 200 entiers compris en 0 et 3 inclus et choisis aléatoirement.

Solution :

```
import random
L = []
for i in range(200):
  L.append(random.randrange(0,4))
# idem comprenhension
L = [random.randrange(0,4) for _ in range(200)]
```

D2. Écrire une fonction de prototype `count(L)` où `L` est une liste d'entiers qui renvoie un dictionnaire dont les clefs sont les éléments de `L` et les valeurs le nombre d'occurrences de cet élément dans `L`.

Solution :

```
def count(L):
  d = {}
  for e in L:
    if e in d:
      d[e] += 1
    else:
      d[e] = 1
  return d
```

D3. Expliquer ce que produit la fonction suivante :

```
def mystery(L):
  s = []
  for i,n in enumerate(L):
    s.append("")
    for k in range(n):
      s[i] += chr(random.randrange(65,91))
  return s
```

Solution : Cette fonction génère une liste de chaînes de caractères en majuscules [A-Z] aléatoirement choisis. Les caractères majuscules de la table ASCII sont codés entre 65 et 91. La longueur de la chaque chaîne d'indice `i` est l'entier `n` situé à d'indice `i` dans la liste L.

D4. Si la liste dont on veut compter les éléments est constituée de chaînes de caractères, la fonction `count` est-elle encore utilisable. Pourquoi?

Solution : Oui, car :

1. une chaîne de caractère est immuable en Python et donc peut devenir la clef d'un dictionnaire,
2. et le code de la fonction `count` ne fait pas d'hypothèse sur le type de `e`.

Cela n'aurait pas été possible si la liste contenait des listes par exemple.

On dit que le code de la fonction `count` est polymorphe, car il s'applique à des plusieurs types de données d'entrée, tant que ceux-ci sont immuables.