INFORMATIQUE

TD n°1 : Listes et boucles

1 Rappels de cours

1.1 Fonctions et procédures

```
def nom_de_la_fonction(parametre1,parametre2):
    # code qui calcule de résultat
    return resultat #return [resultat1,resultat2] ; pour deux résultats
```

```
def nom_de_la_fonction(): #fonction sans paramètre
    # calcul du résultat
    # fonction sans résultat renvoyé (procédure)
```

⚠ ne pas confondre `print` (on affiche et on oublie) et `return` (on peut utiliser le résultat dans la suite).

1.2 Données

- Types simples : `integer` (entier), `float` (réels approchés en virgule flottante), `boolean` (booléens), `char` (caractère).
- Types composés :
  - `list` : liste, modifiable, délimitée par des crochets, séparation par des virgules,
    par ex. : `[1,2,3.0,"a"]` ;
  - `string` : chaîne de caractères, non modifiable, délimitée par ' ou ", pas de séparateur entre les caractères,
    par ex. : `"123a"` ;
  - `tuple` : liste, non modifiable, délimitée par des parenthèses, séparation par des virgules — hors programme *par ex. :* `(1,2,3)`.
  - `array numpy` : cf. TD ultérieur

⚠ **les n éléments d'un type composé sont numérotés de 0 à $n-1$.**

- Opérations usuelles :
  - `x+y` : somme pour les nombres entiers ou flottants, *par ex. :* `1+2` → `3`
    mais concaténation pour les chaînes de caractères ou les listes, *par ex. :* `'1'+'2'` → `'12'`
  - `x*y` : produit pour les nombres entiers ou flottants, mais aussi *par ex. :* `'ar'*2` → `'arar'`
  - division : `x/y` division (résultat réel) ; `x//y` division euclidienne (résultat entier) ;
    `x%y` reste de la division euclidienne, *par ex. :* `6/2` → `3.0` ; `7//2` → `3` ; `12%5` → `2`
  - puissance `x**y`, *par ex. :* `2**5` → `32`
- Conversions (quand c'est possible), *par ex. :*
  `float(3)` → `3.0` ; `int(3.5)` → `3` ; `list("abc")` → `["a","b","c"]` ; `boolean(0)` → `False`
- Manipulation des types composés :
  - nombre d'éléments : `len(x)` (en anglais *length* signifie "longueur") ;
  - appel de l'élément d'indice i (qui est le $i+1$-ème élément) : `x[i]` ;
  - appel du dernier élément : `x[-1]` ou `x[len(x)-1]` ;

  ⚠ « `x[len(x)]` » provoque toujours l'erreur « `out of range` ».

- (*slicing*) extraction d'une tranche de i (inclus) à j (exclu) : `x[i:j]` ;
- (*slicing*) extraction du début (ou de la fin) : `x[:j]` (ou `x[i:]`) ;

- **L'affectation `x=`[$\cdots$] : en général, création d'une variable x qui contient la valeur donnée par [$\cdots$], sauf deux cas particuliers importants :**
  - une variable `x` existait déjà : elle est supprimée puis on revient au cas général.
    Exemple important : `x=x+k` (incrémentation, *i.e.* on ajoute `k` à `x`) se code aussi par : `x+=k`.

  ⚠ faire attention de tenir compte de la portée de la variable, *par exemple :*

```
def f(x):
    y=x
    return y
y=2
z=f(12)
print(y)
```

  → affiche 2 (et non 12).
  - `x=y`, où `y` est une variable d'un type composé : il n'est pas créé de variable `x` mais le contenu de la variable `y` a indifféremment les deux noms `x` et `y` ; toute modification de l'un affecte l'autre ; Analogue avec `x=[y,y]` ou `x=[y]*2`, avec `y` une variable d'un type composé (toute modification d'un parmi `x[0], x[1], y` modifie les autres).
- Affectations simultanées (pour alléger le code) :

```
x,y=a,b  #plusieurs affectations sur un seule ligne
[x,y]=point  #point est un liste de 2 éléments
x,y=y,x   #échange de x et y
```

1.3 Structures conditionnelles

```
if condition:
   #code à exécuter si la condition est réalisée

if condition1:
   #code1, à exécuter si la condition1 est réalisée
elif condition2:
   #code2, à exécuter si la condition1 n'est pas réalisée et la condition2 est
    réalisée
else:
   #code3, à exécuter aucune des conditions précédentes n'est réalisée
```

- `condition1` et `condition2` ne sont pas forcément incompatibles ;
  mais une seule portion de code est exécutée.
- `condition` est le résultat d'un calcul booléen (à valeurs dans `True`, `False`) :
  — Opérations sur tous les types, qui donnent un résultat booléen : `==` ; `!=` ; `>` ; `<` ; `<=` ; `>=`.
  — Opérations entre booléens avec un résultat booléen : `and`/`&` ; `or` ; `not`. *Exemple classique :* `x%2 == 0` $\rightarrow \begin{cases} \texttt{True} & \text{si x est pair} \\ \texttt{False} & \text{sinon} \end{cases}$
  ⇝ l'évaluation paresseuse permet d'éviter des erreurs d'exécution
  par ex. : `i<len(L) and L[i]==12` (deuxième condition non évaluée si la première est fausse)

1.4 Boucles

```
while condition:
   #code à effectuer tant que la condition est réalisée

for i in range(n):
   #code à effectuer n fois
   #si nécessaire on peut utiliser la valeur i qui va de 0 à n-1
```

La boucle `for` est une boucle `while` particulière avec un écriture simplifiée :

```
#équivalent avec while d'une boucle for (maladroit ; à éviter)
i=0
while i<n:
   #code à effectuer n fois
   i=i+1
```

Cas particulier d'une boucle `for` interrompue (dans une fonction) :

```
def mafonction(...):
   for i in range(n):
      if condition:
         ...
         return resultat1 #return termine la fonction donc interrompt la boucle
   return resultat2 #si condition jamais réalisée, toute la boucle est effectuée
#peut remplacer :
def mafonction(...):
   while i<n and not(condition):
      ...
if condition:
   return resultat1
else:
   return resultat2
```

⚠ Dans une boucle `while`, il faut que les paramètres de la condition soient modifiés par le code de la boucle ; sinon la boucle ne se termine jamais.

⚠ il est vivement déconseillé de modifier la variable `i` de la boucle `for` à l'intérieur de la boucle ;

- Extension des itérateurs pour les boucles `for` :
  - `i in range(n)` : `i` parcourt les entiers de 0 à $n-1$;
  - `i in range(n,p)` : `i` parcourt les entiers de n à $p-1$;
  - `i in range(n,p,q)` : valeurs de n (compris) à p (non compris) avec un pas de q *i.e.* `i` parcourt les entiers n, $n+q$, $n+2q$ ...$n+kq$, avec $n+kq < p \leqslant n+(k+1)q$.
  - `l in L` : `l` parcourt tous les éléments de la liste `L` (qui ne sont pas forcément des entiers).

```
for i in range(len(L)):
   #calcul sur L[i] (mais pas sur i)
#revient au même que :
for l in L:
   #calcul sur l
```

1.5 Construction d'une liste ou d'un tableau

- À l'aide d'une boucle avec la méthode append :

```
L=[] # initialisation
for i in range(n) :
   L.append(fonction(i)) # ajoute le résultat de fonction(i) à la fin de L
```

- À l'aide d'une boucle par concaténation :

```
L=[] # initialisation
for i in range(n) :
   L=L+[fonction(i)]  #L=[fonction(i)]+L permet de créer L à partir de la fin
```

- Avec l'écriture en compréhension :

```
L=[fonction(i) for i in range(n)]
```

- Cas particulier d'une « vraie » copie de liste :

```
#à l'aide d'une boucle
copie=[]
for i in range(len(original)):
     copie.append(original[i])
     #ou copie=copie+[original[i]]
#ou en compréhension
copie=[l for l in original]
#voire (à expliquer en commentaire à chaque fois)
copie=original[:]
#ou avec une fonction dédiée
import copy
copie=copy.deepcopy(original)
```

Pour copier une liste de liste, seul `deepcopy` fonctionne bien.

1.6 Algorithmes classiques de recherche dans une liste `L`

- **Recherche si l'un des éléments de la liste vérifie une condition donnée :**

```
def recherche1(L):
   for j in L:
      if condition(j)==True:
         return True #boucle interrompue (while possible aussi)
   return False
```

Exemples : tester la présence d'un élément donné dans une liste, comparer deux listes (revient à rechercher s'il existe deux élements homologues différents)

- **Recherche d'un élément optimal (d'un certain point de vue) dans une liste :**

```
resultat=L[0] # ou initialiser avec le pire résultat possible
for j in L:
   if j meilleurque resultat:
      resultat=j
```

Exemples : maximum, minimum.

- **Résultat obtenu par agrégation de tous, ou de certains éléments d'une liste :**

```
resultat=initialisation #resultat si la liste était vide
for [condition sur j] :
   resultat=[fonction de resultat et de j]
```

Exemples : somme, produit, moyenne, liste ou nombre des éléments vérifiant une condition donnée (alors `resultat=calcul(resultat,j)` est de la forme `if condition(j)==True: resultat=...`).

2 Exercices

EXERCICE. —

Q1 (Création de liste). Écrire deux façons différentes (avec `append` et en compréhension) une fonction `zero_un(n)` qui renvoie la liste formée de `n` nombres qui valent alternativement `0` puis `1`.
Par exemple : `zero_un(7)`→`[0,1,0,1,0,1,0]`

Q2 (Test d'appartenance). Écrire une fonction `est_element(L,a)` qui renvoie `True` si `a` est élément de la liste `L`, et `False` sinon.

Q3 (Compteur).
Écrire une fonction `NB(L,M)` qui renvoie le nombre d'éléments de `L` qui appartiennent à `M`.

EXERCICE. — (construction de listes de listes)

Q4 Écrire une fonction `listenulle(n,p)` qui renvoie la liste `[l1,l2,...,ln]` où chaque `li` est une liste de p zéros.
Par exemple : `listenulle(3,2)` → `[[0,0],[0,0],[0,0]]`
On programmera cette fonction pour qu'on puisse l'utiliser afin de créer un tableau de taille 11×11 dont toutes les cases sont remplies de zéros sauf celle du centre qui vaut 1 (indiquer comment).

Q5 Écrire une fonction `L2(l)`, d'argument une liste `l` d'entiers de longueur `n`, qui renvoie la liste `[l0,l1,...,ln-1]` où chaque `li` est une liste de `l[i]` zéros.
Par exemple : `L2([3,2,4])` → `[[0,0,0],[0,0],[0,0,0,0]]`.

Q6 Écrire une fonction `A(n,a,b)`, où `n` est un entier positif, qui renvoie le tableau $n \times n$ dont toutes les cases contiennent `0` sauf celles de la diagonale qui valent `1` et celles de la première ligne et de la première colonne qui valent alternativement `b` puis `a`.
Par exemple : `A(5,8,3)` → `[[1,3,8,3,8],[3,1,0,0,0],[8,0,1,0,0],[3,0,0,1,0],[8,0,0,0,1]]`

1	3	8	3	8
3	1	0	0	0
8	0	1	0	0
3	0	0	1	0
8	0	0	0	1

Q7 Écrire une fonction `T(L,n)`, où `L=[a,b,c]` est une liste de trois nombres et `n` est un entier supérieur à 2, qui renvoie une matrice carrée d'ordre `n` avec `b` sur la diagonale, des `a` au dessus de la diagonale et des `c` en dessous de la diagonale.

EXERCICE. — (Suite récurrentes). Soit la suite définie par $u_0 \in \mathbb{N}^*$ et : $\forall n \geqslant 0$, $u_{n+1} = \begin{cases} 3u_n + 1 & \text{si } u_n \text{ impair} \\ \frac{u_n}{2} & \text{si } u_n \text{ pair.} \end{cases}$

Q8 Écrire une fonction `suite(u0,n)` qui calcule u_n.

Q9 Écrire une fonction `somme(u0,n)` qui calcule $u_0 + \cdots + u_n$.

Q10 Écrire une fonction `prod(u0,n)` qui calcule le produit des termes u_i de la suite dont la valeur est paire et tels que $0 \leqslant i \leqslant n$.

Q11 On définit la suite (u_n) par $u_0 = u_1 = 1$ et $u_{n+2} = u_{n+1} + u_n$.
Écrire une fonction `fibo(n)` qui calcule u_n.

EXERCICE. —

Q12 (maximum) Les sommes partielles d'une liste `L` d'entiers sont les $S_k = \sum_{i=0}^{k} L[i]$ $(0 \leqslant k <$`len(L)`$)$.
Écrire une fonction `max_SP(L)` qui renvoie la plus grande somme partielle de la liste `L`.

Q13 Écrire une fonction `max_somme_ligne(L)`, où `L` est une liste de listes d'entiers, qui renvoie la plus grande somme des éléments d'une liste `L[i]`.
Par exemple : `max_somme_ligne([[0,0,0,9,0,0],[3,7,0],[2,1,1,1]]` → `10`.

3 Exercices plus difficiles

EXERCICE. —

Q14 Écrire une fonction `bissextile(n)` qui renvoie `True` si l'année `n` est bissextile (*i.e.* divisible par 400 ou divisible par 4 mais pas par 100) ou `False` sinon.

Q15 Écrire une fonction `jourannee(jour, mois, annee)` qui renvoie le nombre de jours écoulés depuis le début de l'année `annee` jusqu'à la date (incluse) donnée par `[jour, mois, annee]`.
Par exemple : `jourannee([2, 3, 2004])→ 62 jourannee([2, 3, 2003])→ 61`

Q16 On a compté 1446 pleines lunes entre le 15 janvier 1900 et le 14 novembre 2016. Programmer un script qui calcule la période de révolution de la Lune.

EXERCICE. — On appelle « renversé » d'un entier le nombre obtenu en écrivant les chiffres de l'entier de départ à l'envers. Par exemple le renversé de 563 est 365. Un entier palindrome est un entier qui est égal à son renversé, par exemple 121 ou 1331.

Q17 Écrire une fonction `estPalindrome(n)` qui renvoie `True` si `n` est un palindrome ou `False` sinon. On pourra remarquer que : `list(str(123))→ ['1','2','3']`

Q18 Écrire une fonction `verlan(n)` qui renvoie l'entier obtenu en écrivant les chiffres de l'entier `n` à l'envers. *Par exemple :* `verlan(563)→ 365`

Q19 On note $r(n)$ la somme de n et de son renversé. Écrire une fonction `pal(n,N)` qui renvoie la liste `[n,r(n),r(r(n)),...,r`^{k}`(n)]` où k est le plus petit entier inférieur ou égal à `N` tel que `r`^{k}`(n)` soit un palindrome, s'il en existe un, et qui renvoie `False` s'il n'en existe pas.
Par exemple : `pal(64,50)→[64,110,121]` et `pal(98,20)→ False`

INFORMATIQUE

Corrigé du TD n°1

```python
#####Q1###
def zero_un(n):
    L=[]
    for i in range(n):
        L.append(i%2)
    return L

#ou
def zero_un1(n):
    L=[]
    for i in range(n):
        L=L+[i%2]
    return L

#ou
def zero_un2(n):
    L=[i%2 for i in range(0,n)]
    return L

#####Q2###
def est_element1(L,a):
    """ parcours par les indices"""
    for i in range(len(L)):
        if L[i]==a:
            return True  #boucle interrompue
    return False
def est_element2(L,a):
    """ parcours par les elements"""
    for x in L:
        if x==a:
            return True  #boucle interrompue
    return False

#version (plus maladroite) avec while, et sans indexation par une liste
def est_element(L,a):
    i=0
    l=len(L)-1 #index du dernier élément de L --- car numérotation à partir de 0
    while L[i] != a and i<l:
        i+=1
    if L[i]==a:
        return True
    else:
        return False
#les constantes True et False s'écrivent avec un majuscule

```

```python

#####Q3###
def NB(L,M):
    compteur=0
    for x in L:
        if est_element(M,x)==True:
            compteur=compteur+1 # ou compteur+=1
    return compteur

#####Q4###
#Premier essai (incorrect)
def liste_nulle(n,p):
    """ avec append"""
    Ligne=[]
    for i in range(p):
        Ligne.append(0)
    Tableau=[]
    for i in range(n):
        Tableau.append(Ligne)
    return Tableau

#plutôt
T=liste_nulle(3,4)
T[0][2]=5 #modifie toutes les lignes

def liste_nulle(n,p):
    """ avec append"""
    Tableau=[]
    for i in range(n):
        Ligne=[]
        for i in range(p):
            Ligne.append(0)
        Tableau.append(Ligne)
    return Tableau

#ou
def liste_nulle1(n,p):
    """ par comprehension """
    Tableau=[[0 for i in range(p)] for i range(n)]


#####Q5###
def L2(l):
    """ par comprehension """
    Tableau=[[0 for i in range(l[i])] for i in range(len(l))]

#ou
def L2(l):
    """ avec append"""
    Tableau=[]
    for i in range(len(l)):
        Ligne=[]
        for i in range(l[i]):
            Ligne.append(0)
```

```
        Tableau.append(Ligne)
    return Tableau

#####Q6###
def A(n,a,b):
    T1=liste_nulle(n,n)
    for i in range(n):
        T1[i][i]=1
    for i in range(1,n):
        if i%2==0:
            T1[0][i],T1[i][0]=a,a
        else:
            T1[0][i],T1[i][0]=b,b
    return T1

#####Q7###
def T(L,n):
    a,b,c=L
    T1=liste_nulle(n,n)
    for i in range(n):
        for j in range(n): # parcours de tous les couples (i,j)
            if i==j:
                T1[i][j]=b
            elif i<j:
                T1[i][j]=a
            else:
                T1[i][j]=c
    return T1

#####Q8###
def suite(u0, n):
    u=u0
    for i in range(n):
        if u%2==0:
            # a%b est le reste de de la division euclidienne de a par b
            u=u/2
        else:
            u=3*u+1
    return u
# ne pas oublier le ":" dans les structures de programmation
# attention à l'indentation

#####Q9###
def somme(u0,n):
    """ version avec appel de suite"""
    S=0
    for i in range(n+1):
        S=S+suite(u0,i) #Complexite O(i)
    return S #Complexite O(n**2)

#mieux :
def somme(u0,n):
    u,S=u0,u0
    for i in range(n):
```

```
        if u%2==0:
            u=u/2
        else:
            u=3*u+1
        S+=u #écriture abrégée de S=S+u (inspirée du langage C++)
    return S
#meilleure complexité --- en O(n) --- que l'utilisation de suite(u0,n)

#####Q10###
def prod(u0,n):
    u=u0
    P=1
    for i in range(n+1):
        if u%2==0:
            P=P*u
        if u%2==0:
            u=u//2
        else:
            u=3*u+1
    return P
#meilleure complexité que l'utilisation de suite(u0,n)

#####Q11###
def fibo(n):
    u,v=1,1 #deux premiers termes
    for i in range(n-1):
        w=v
        v=u+v
        u=w #on peut remplacer ces trois lignes par l'affectation simultanée u,v=v
    ,u+v
    return v
#meilleure complexité en espace que de stocker tous les termes de le suite

#####Q12###
def max_SP(L):
    S=0 #initialisation de la somme partielle
    M=L[0] #initialisation du max des sommes partielles de 0 à i
    for i in L:
        S=S+i
        if M<S:
            M=S
    return M

#ou
def SP(L):
    """ renvoie le max des sommes partielles"""
    S,max=L[0],L[0]
    for i in range(1,len(L)):
        S=S+L[i]
        if S>max:
            max=S
    return max

#####Q13###
```

```python
def somme_ligne(l):
    S=0
    for i in l:
        S+=i
    return S
def max_somme_ligne(L):
    M=somme_ligne(L[0])
    # initialisation du maximum
        # plutot que 0, au cas ou il y aurait des nombres négatifs dans les listes
    for l in L:
       S=somme_ligne(l)
       if S>M:
           M=S
    return M

#ou
def somme(l):
    """ renvoie la somme d'une liste"""
    S=0
    for i in range(len(l)):
        S=S+l[i]
    return S
def max_somme_ligne(L):
    """parcours par elements"""
    max=somme(L[0])
    for l in (L):
        if somme(l)>max:
            max=somme(l)
    return max

###Q14###
def bissextile(n):
    if n%400==0:
        res=True
    elif n%4==0 and n%100!=0:
        res=True
    else:
        res=False
    return res

###Q15###
def jourannee(jour,mois,annee):
    """jour numerote de 1 a 31, mois de 1 a 12"""
    L=[31,28,31,30,31,30,31,31,30,31,30,31]
    Lb=[31,29,31,30,31,30,31,31,30,31,30,31]
    N=0
    if bissextile(annee)==True:
        for i in range(0,mois-1):
            N=N+Lb[i]
    else:
        for i in range(0,mois-1):
            N=N+L[i]
    N=N+jour
    return N
```

```
261
262 ###Q16###
263 # on suppose qu'une pleine lune a ete obervee le 15 janvier 1900 et le 14
264 # novembre 2016
265 # entre le premier janvier 1900 et le premier 31 decembre 2015
266 N=0
267 for i in range(1900,2016):
268     if bissextile(i):
269         N=N+366
270     else:
271         N=N+365
272
273 # entre le 15 janvier 1900 et le premier 31 decembre 2015
274 N=N-14
275 # entre le 15 janvier 1900 et le premier 13 novembre 2016
276 N=N+jourannee(13,11,2016)
277 print(N/1445)
278
279 ###Q17###
280 def est_palindrome(n):
281     s=str(n) # conversion en chaine de caracteres
282     res=True
283     for i in range(len(s)):
284         if s[i]!=s[len(s)-1-i]:
285             res=False
286     return res
287
288 ###Q18###
289 def verlan(n):
290     s=str(n)
291     v='' #chaine vide
292     for i in range(len(s)):
293         v=v+s[len(s)-1-i]
294     return int(v)
295 def verlan2(n):
296     s=str(n)
297     v='' #chaine vide
298     for i in range(len(s)-1,-1,-1): # boucle avec pas de -1
299     # debut compris, fin exclue, pas entier
300         v=v+s[i]
301     return int(v)
302
303 ###Q19###
304 def r(n):
305     return n+verlan(n)
306 def pal(n,N):
307     L=[]
308     for i in range(N):
309         L.append(n)
310         if est_palindrome(n)==True:
311             return L
312         n=r(n)
313     return False
314
```

```
###Q20###
def suivant(L,p):
    n=len(L)
    i=(p+1)
    if i==n:
        i=0 # positionnement en cercle
    while L[i]==True:
        i=(i+1)
        if i==n:
            i=0
    return i
def reste(n):
    L=[False for i in range(n)]
    p=0
    for i in range(n-1):
        i=suivant(L,p)# celui qui disparait
        L[i]=True
        p=suivant(L,i)# celui qui reste
    return positionTrue(L)
def positionTrue(L):
    for i in range(len(L)):
        if L[i]==False:
            return i

for i in range(100):
    print(i,reste(i))

###Q21###

###Q22###
def un_tour(L,k):
    Tour=[]
    for j in range(k,len(L)-k):
        Tour.append(L[k][j])
    for i in range(k+1,len(L)-k):
        Tour.append(L[i][len(L)-k-1])
    for j in range(len(L)-k-2,k-1,-1):
        Tour.append(L[len(L)-k-1][j])
    for i in range(len(L)-k-2,k,-1):
        Tour.append(L[i][k])
    return Tour
def escargot(L):
    E=[]
    for k in range((len(L)+1)//2):
        E=E+un_tour(L,k)
    return E

L=[[i*5+j for j in range(5)] for i in range(5)]
"""
Out[3]:
[[0, 1, 2, 3, 4],
 [5, 6, 7, 8, 9],
 [10, 11, 12, 13, 14],
 [15, 16, 17, 18, 19],
```

```
 [20, 21, 22, 23, 24]]
"""
In [10]: un_tour(L,0)
Out[10]: [0, 1, 2, 3, 4, 9, 14, 19, 24, 23, 22, 21, 20, 15, 10, 5]
In [21]: print(escargot(L))
[0, 1, 2, 3, 4, 9, 14, 19, 24, 23, 22, 21, 20, 15, 10, 5, 6, 7, 8, 13, 18, 17, 16,
    11, 12]
```