

Fiche 5 - Chaînes de caractères et tableaux

I. CHAÎNES DE CARACTÈRES

Les chaînes de caractères sont délimitées par des apostrophes ou des guillemets.

→ Les fonctions spécifiques :

Notation	Explication
'...'+'...' str(x)	Concaténation de deux chaînes Convertit l'expression x en chaîne de caractères
list(c)	Donne la liste des caractères de la chaîne c
".join(L)	Concatène les chaînes, où L est un liste de chaînes
len(c)	Donne la longueur
a in c	Teste si la chaîne a est incluse dans la chaîne c
c[0],c[1], ...	Les caractères de gauche à droite
c[-1],c[-2], ...	Les caractères de droite à gauche
c[-1]	Le dernier caractère ; c[len(c)-1]
c[i:j]	Les caractères donc la position varie sur [i,j[
c[i:j:k]	Les caractères donc la position varie de k en k entre la ième et le j-lème
c[i:]	La sous-chaîne commençant au ième caractère
c[-i:]	La sous-chaîne des i derniers caractères
c[:i]	La sous-chaîne des i premiers caractères (de 0 à i-1)

```
>>> str(3.14)
'3.14'
>>> a='bon';b='week-end'
>>> type(a)
<class 'str'>
>>> c=a+' '+b;c
'bon week-end'
>>> L=list(c)
>>> L=list(c);L
['b', 'o', 'n', ' ', 'w', 'e', 'e', 'k', '-', ...
>>> d=''.join(L);d
'bon week-end'
>>> len(c)
12
>>> 'k' in c
True
>>> c[0];c[1];c[11]
'b'
'o'
'd'
```

```
>>> c[-1];c[-11];c[-12]
'd'
'o'
'b'
>>> c[2:5]
'n w' # lettre de position 2,3 et 4
>>> c[0:12:2] # une lettre sur deux
'bnwe-n'
>>> c[:4]
'bon '
>>> c[4:]
'week-end'
>>> c[-8:]
'week-end'
```

→ Des attributs

Ces fonctions sont proposées lorsque l'on met un point après la variable. Une documentation plus complète est donnée dans *l'inspecteurs d'objets*, en particulier les arguments optionnels.

Notation	Explication
c.replace(a,b)	Retourne une liste où les occurrences de la chaîne a sont remplacées par la chaîne b dans c. Un troisième argument, optionnel, n, permet de limiter les modifications aux n premières occurrences.
c.count(ch)	Dénombre les occurrences de la chaîne ch dans la chaîne c
c.index(ch)	Donne la position de la première occurrence de la chaîne ch dans la chaîne c
c.find(ch)	Idem ; mais si la chaîne ch n'apparaît pas dans c alors retourne -1.

Table 5.1: Attributs de chaînes

```
>>> d=c.replace('e','E',2)
>>> d;c
'bon wEEk-end'
'bon week-end'
>>> c.count('e')
3
>>> c.index('ee')
5
>>> c.find('eed')
-1
```

Attention ! Dans le cas des chaînes de caractères, les attributs ne modifient pas la chaîne sujet mais renvoie une nouvelle chaîne.

Remarque : Il est parfois utile de définir une chaîne de caractères

tères tenant compte de la valeur d'une variable ; par exemple, concernant le nom d'une figure :

```
>>> n=12
>>> nom='figure pour n = '+str(n)+' .pdf';nom;
'figure pour n = 12.pdf'
```

Attention ! Lorsqu'une chaîne de caractère contient une apostrophe, il est préférable d'utiliser les guillemets.

```
>>> "aujourd'hui"
"aujourd'hui"
>>> 'aujourd\'hui'
"aujourd'hui"
```

II. LISTES

Une liste se note avec des crochets. Ses éléments sont séparés par une virgule et ordonnés. Les éléments peuvent être de types différents.

→ Les fonctions spécifiques sont identiques à celles pour les chaînes :

Notation	Explication
list(ite)	Convertit en liste un expression itérable : chaîne, intervalle, tableau ligne, uplet
[]	Liste vide
L+M	Concaténation de deux listes
len(L)	Donne la longueur
a in L	Teste l'appartenance
L[0],L[1], ...	Les éléments de gauche à droite
L[-1],L[-2], ...	Les éléments de droite à gauche
L[-1]	Le dernier élément ; L[len(L)-1]
L[i:j]	Les éléments de position dans $[[i, j - 1]]$ L[i:j:k], L[i:], L[-i:], L[:i]

Table 5.2: Fonctions sur les listes

```
>>> L=['abc',2,4.1]+[[1],1j];L
['abc', 2, 4.1, [1], 1j]
>>> len(L)
5
>>> 12 in L
False
>>> L[0];L[-1];L[1:3]
'abc'
1j
[2, 4.1]
```

Les itérables

```
>>> list(range(2,13,3))
[2, 5, 8, 11]
>>> list('azerty')
['a', 'z', 'e', 'r', 't', 'y']
>>> list((2,3,4,5))
[2, 3, 4, 5]
```

→ Des attributs

Notation	Explication
L.append(a)	Rajoute l'élément "a" à la liste L ; L=L+[a]
L.extend(M)	Rajoute à la liste L les éléments de la liste M ; L=L+M
L.insert(i,a)	Insère l'élément a à la position i
L.remove(a)	Retire la première occurrence de l'élément a
L.pop(i)	Retire et renvoie le ième élément.
L.count(a)	Dénombre les occurrences de l'élément a
L.index(a)	Donne la position de la première occurrence de l'élément a
L.sort()	Trie la liste
L.reverse()	Inverse l'ordre des éléments

Table 5.3: Attributs de listes

```
>>> L=[3,1,5,12]
>>> L.append(0);L
[3, 1, 5, 12, 0]
>>> L.extend([2,8]);L
[3, 1, 5, 12, 0, 2, 8]
>>> L.insert(4,5);L
[3, 1, 5, 12, 5, 0, 2, 8]
>>> L.remove(12);L
[3, 1, 5, 5, 0, 2, 8]
>>> L.pop(4);L
0
[3, 1, 5, 5, 2, 8]
>>> L.index(5)
2
>>> L.count(5)
2
>>> L.sort();L
[1, 2, 3, 5, 5, 8]
>>> L.reverse();L
[8, 5, 5, 3, 2, 1]
```

Attention ! Dans le cas des listes, les attributs modifient la liste sujet.

→ La fonction enumerate

Un objet itérable est un objet dont les éléments qui le constituent peuvent être listés ; par exemple : une liste, une chaîne de caractère, un tableau unidimensionnel.

L'instruction "for i,e in enumerate(L)" permet de parcourir l'objet itérable L où i parcourt les positions (en partant de 0) et e parcourt les éléments correspondants.

```
def pos_paire(L):
    '''fonction qui retourne la liste
    des positions des elements pairs'''
    p=[]
    for i,e in enumerate(L):
        if e%2==0:
            p.append(i)
    return p
```

Définition par compréhension

Il est possible de définir une liste par compréhension grâce à l'introduction de l'instruction for :

```
>>> [i**2 for i in range(5,0,-1)]
[25, 16, 9, 4, 1]
>>> I=range(1,12,2)
>>> [i%3==0 for i in I]
[False, True, False, False, True, False]
>>> [i for i in range(21) if i%3==0]
[0, 3, 6, 9, 12, 15, 18]
>>> [str(i)+'_'+str(j) for i in range(1,4)
      for j in range(i,4)]
['1_1', '1_2', '1_3', '2_2', '2_3', '3_3']
```

⇒ Exemple donnant l'extraction des voyelles d'une chaîne de caractère :

```
>>> mot="Bon_week-end"
>>> voyelles="aeiouy"
>>> [i for i in mot if i in voyelles]
['o', 'e', 'e', 'e']
```

⇒ Exemple de calcul de sommes :

On charge le module numpy :

```
import numpy as np
```

Notation	Explication
np.sum(L)	Somme des éléments de L <i>nombre ou expressions booléennes</i>
np.prod(L)	Produit des éléments de L

```
>>> np.sum([1,2,3,4])
10
>>> np.sum([True, False, False, True])
2
```

Calcul de $\sum_{k=1}^{12} \frac{1}{k^2}$

```
>>> np.sum([1/k**2 for k in range(1,1001)])
1.6439345666815615
>>> pi**2/6
1.6449340668482264
```

Calcul de $\sum_{1 \leq i < j \leq 12} \frac{1}{i^2 + j^2}$

```
>>> np.sum([1/(i**2+j**2) for i in range(1,13)
            for j in range(i,13)])
2.0023372508030217
```

Alias - copie de liste

L'affectation T=L créé un alias de la liste L. Toute modification sur T ou L est automatiquement effectuée sur l'autre liste.

```
>>> L=[1,3,6,9]
>>> T=L
>>> T
[1, 3, 6, 9]
>>> T.append(12)
>>> T
[1, 3, 6, 9, 12]
>>> L
[1, 3, 6, 9, 12]
>>> L[0]='ok'
>>> L
['ok', 3, 6, 9, 12]
>>> T
['ok', 3, 6, 9, 12]
```

Cependant, si l'une des deux listes est redéfinie, le lien entre les deux est rompu :

```
>>> T=[1,2]
>>> T
[1, 2]
>>> L
['ok', 3, 6, 9, 12]
```

→ Pour faire une copie d'une liste de nombres, il suffit d'écrire T=1*L. Cette fois les deux listes ne sont pas liées.

```
>>> L=[1,3,5,6]
>>> T=1*L
>>> L
[1, 3, 5, 6]
>>> T
[1, 3, 5, 6]
>>> T[0]=12
>>> L
[1, 3, 5, 6]
>>> T
[12, 3, 5, 6]
```

Autre méthode, en particulier lorsque la liste ne contient pas que des nombres :

```
>>> T=[x for x in L]
>>> T
[1, 3, 5, 6]
>>> T[0]=12
>>> L
[1, 3, 5, 6]
>>> T
[12, 3, 5, 6]
```

III. POLYNÔMES

Le module numpy contient la classe poly1d qui est un *type* polynômial.

```
>>> import numpy as np
>>> P=np.poly1d([1,2,3,4])
>>> print(P) # defini par ses coeff
      3      2
1 x + 2 x + 3 x + 4
>>> Q=np.poly1d([0,0,0,1],True)
>>> print(P) # defini par ses racines
      3      2
1 x + 2 x + 3 x + 4
>>> type(P)
<class 'numpy.lib.polynomial.poly1d'>
```

Les fonctions suivantes, du **module numpy** existent aussi sous la forme d'attributs :

Notation	Explication
poly1d(L)	Polynôme dont les coefficients sont donnés dans le liste L dans l'ordre des puissances décroissantes
P[i]	Coefficient de X^i
P(a)	Evaluation de P en a
P.coeffs	Liste des coefficients
len(P)	Dégré, voir P.order
polyder(P)	Polynôme dérivé, voir P.deriv()
polyint(P)	Primitive de P nulle en 0 voir P.integ()
roots(P)	Racines complexes de P
poly1d(L,True)	Polynôme dont les racines sont donnés dans le liste L
q,r=A/B	Quotient et reste de la division euclidienne de A par B voir q,r=polydiv(A,B)

```
>>> Q=np.poly1d([1,2,2,-1],True)
>>> print(Q)
      4      3      2
1 x - 4 x + 3 x + 4 x - 4
>>> len(Q)
4
>>> Q.coeffs
array([ 1, -4,  3,  4, -4])
>>> roots(Q)
array([-1.          ,  2.00000005,
  1.99999995,  1.          ])
>>> Q(2)
0
>>> Q[2]
3
>>> np.polyder(Q)
poly1d([ 4, -12,  6,  4])
>>> np.polyint(Q)
poly1d([ 0.2, -1. ,  1. ,  2. , -4. ,  0. ])
```

IV. TABLEAUX

Le type array est la représentation classique des tableaux : la liste de ses lignes.

```
>>> T=np.array([[1,2,3],[4,5,6]])
>>> type(T)
<class 'numpy.ndarray'>
>>> T
array([[1, 2, 3],
       [4, 5, 6]])
```

→ définition par compréhension

```
>>> f=lambda x,y:abs(x-y)
>>> B=np.array([[f(i,j) for j in range(4)]
               for i in range(3)])
>>> B
array([[0, 1, 2, 3],
       [1, 0, 1, 2],
       [2, 1, 0, 1]])
```

➤ On peut aussi utiliser la fonction np.fromfunction. L'argument optionnel dtype permet de proposer un type souhaité pour les éléments du tableau :

```
>>> C=np.fromfunction(f,(3,4),dtype=int);C
array([[0, 1, 2, 3],
       [1, 0, 1, 2],
       [2, 1, 0, 1]])
```

→ Eléments basiques du **module numpy** :

Notation	Explication
shape(A)	Taille de la matrice A : (l, c)
shape(A)[0]	Nombre de lignes, voir len(A)
reshape(A,(l,c))	Reformate le tableau a
size(A)	Nombres de coefficients
rank(A)	Dimension de la matrice A
A[:,i]	La ième colonne de A
A[i,:]	La ième ligne de A
A[i,j]	Le coefficient (i,j)
trace(A)	Trace de A
transpose(A)	Tranposée de A
dot(A,B)	Produit matriciel
A*B	Produit termes à termes

Attention ! Les opérations *, **, / se font termes à termes.

```
>>> np.shape(T)
(2, 3)
>>> np.shape(T)[0]
2
>>> np.reshape(T,(3,2))
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```

>>> np.rank(T)
2
>>> T[0,0]
1
>>> T[:,2]
array([3, 6])
>>> T[:,0]=array([7,8])
>>> T
array([[7, 2, 3],
       [8, 5, 6]])
>>> np.trace(T)
12
>>> np.transpose(T)
array([[7, 8],
       [2, 5],
       [3, 6]])
>>> T*T
array([[49, 4, 9],
       [64, 25, 36]])
>>> dot(T,array([1,1,1]))
array([12, 19])
>>> dot(T,array([[1],[1],[1]]))
array([[12],
       [19]])

```

Remarque : PYTHON accepte une certaine latitude dans l'écriture des vecteurs.

→ Matrices particulières :

Notation	Explication
<code>zeros((n,m))</code>	Matrice nulle de taille (n,m)
<code>eye(n,m)</code>	Matrice de zéros, excepté sur la diagonale principale qui contient des 1
<code>identity(n)</code>	Matrice identité I_n voir <code>np.eye(n,n)</code> ou <code>np.eye(n)</code>
<code>ones((n,m))</code>	Matrice de 1 de taille (n,m)
<code>diag(v)</code>	Matrice diagonale, de diagonale les coefficients du vecteur (ou liste) v
<code>diag(v,i)</code>	Matrice carrée dont la i ème diagonale contient les coefficients de v

```

>>> np.zeros((2,4))
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.eye(2,4)
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.]])
>>> np.identity(2)
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> np.ones((1,5))
array([[ 1.,  1.,  1.,  1.,  1.]])
>>> np.diag([2,1],-1)
array([[0, 0, 0],
       [2, 0, 0],
       [0, 1, 0]])

```

→ Certaines des fonctions suivantes sont dans le **sous-module linalg du module numpy** :

```
>>> import numpy.linalg as LA
```

Notation	Explication
<code>solve(A,V)</code>	Résout $AU = V$ d'inconnue U où V est un vecteur colonne ou compatible avec A
<code>inv(A)</code>	Inverse de A
<code>inner(U,V)</code>	Produit scalaire des vecteurs lignes U et V
<code>det(A)</code>	Déterminant
<code>matrix_power(A,n)</code>	Puissance nième de A
<code>eig(A)</code>	Elements propres de A
<code>matrix_rank(A)</code>	Rang

```

>>> A=array([[1,0,2],[-1,1,1],[0,1,-2]]);A
array([[ 1,  0,  2],
       [-1,  1,  1],
       [ 0,  1, -2]])
>>> det(A)
-4.9999999999999991
>>> dot(A,A) # produit de matrices
array([[ 1,  2, -2],
       [-2,  2, -3],
       [-1, -1,  5]])
>>> matrix_power(A,2)
array([[ 1,  2, -2],
       [-2,  2, -3],
       [-1, -1,  5]])
>>> A*A # produit termes a termes
array([[1, 0, 4],
       [1, 1, 1],
       [0, 1, 4]])
>>> B=inv(A)
>>> dot(A,B)
array([[ 1.00000000e+00,  0.00000000e+00,
 0.00000000e+00],
       [-5.55111512e-17,  1.00000000e+00,
 5.55111512e-17],
       [-1.11022302e-16, -1.11022302e-16,
 1.00000000e+00]])

```

Attention ! L'inverse de A est calculée numériquement et donc avec une approximation. Il est normal que le produit d'une matrice et de son inverse ne donne pas exactement la matrice identité.

Attention ! La problématique de l'alias/copie de tableaux est la même que pour les listes.

Remarque : Les opérations entre un nombre et un tableau sont effectuées sur chaque coefficient.

```
>>> A=zeros((2,4))+3;A
array([[ 3.,  3.,  3.,  3.],
       [ 3.,  3.,  3.,  3.]])
>>> B=3*ones((2,4));B
array([[ 3.,  3.,  3.,  3.],
       [ 3.,  3.,  3.,  3.]])
```

Tableaux aléatoires

Chargeons le **sous-module random du module numpy** :

```
import numpy.random as rd
```

On considère deux fonctions :

Notation	Explication
rand()	Nombre aléatoire sur [0,1[
rand(n)	tableau ligne de n nombres aléatoires sur [0,1[
rand(n,m)	tableau de n lignes et m colonnes de nombres aléatoires sur [0,1[
randint(n)	Entier aléatoire sur $[0, n[$
randint(n,m)	Entier aléatoire sur $[n, m[$
	argument optionnel <code>size=k</code> pour un tableau ligne de k entiers et <code>size=(i,j)</code> pour un tableau de taille (i,j)

```
>>> a=rd.rand(4);a
array([ 0.61824229,  0.85338083,
        0.17162161,  0.43392505])
>>> np.sum(a>(0.5*np.ones((1,4))))
2

# tableau de 5 entiers aleatoires sur [0,12]
>>> [round(12*rd.rand()) for i in range(5)]
[7, 11, 10, 8, 3]
>>> rd.randint(0,13,size=5)
array([0, 1, 9, 2, 5])
>>> [int(13*rd.rand()) for i in range(5)]
[0, 3, 0, 5, 9]

>>> b=rd.randint(2,size=(2,4));a
array([[1, 1, 1, 1],
       [1, 0, 1, 1]])
>>> np.sum(b==np.ones((2,4)))
7 # nombre de 1
>>> np.sum(b)
7
```

Autre type de tableau : matrix

Il existe un type `matrix` dans le **module numpy**.

```
>>> A=rd.randint(2,size=(3,3))
>>> A
array([[1, 0, 1],
       [1, 0, 0],
       [0, 1, 0]])
>>> B=np.matrix(A);B
matrix([[1, 0, 1],
        [1, 0, 0],
        [0, 1, 0]])
>>> type(B)
<class 'numpy.matrixlib.defmatrix.matrix'>
```

Les fonctions précédentes s'appliquent à cet objet. Les opérations `*`, `**` sont, cette fois, des opérations sur les matrices :

```
>>> A*A
array([[1, 0, 1],
       [1, 0, 0],
       [0, 1, 0]])
>>> B*B
matrix([[1, 1, 1],
        [1, 0, 1],
        [1, 0, 0]])
>>> np.dot(A,A)
array([[1, 1, 1],
       [1, 0, 1],
       [1, 0, 0]])
```