Structures séquentielles de données

Développeur Python

Sommaire

1. Généralités sur les séquences.
2. Les listes.
3. Les chaînes de caractères.
4. Les t-uples.

1. Généralités sur les séquences.

1. Généralités sur les séquences.

Motivation première

- Réunir au sein d'une même variable plusieurs valeurs différentes.

- L'objectif étant d'optimiser certaines opérations comme la recherche d'un élément, le tri de ces valeurs, le calcul de leur maximum, etc.

1. Généralités sur les séquences.

Remarque

- Cette “réunion“ de valeurs peut se faire de plusieurs façons différentes :

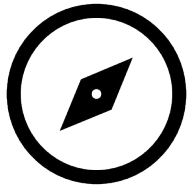
    - Structures séquentielles
    - Ensembles (*cf.* cours 1ALGO)
    - Dictionnaires (*cf.* cours 1ALGO)
    - Classes (*cf.* cours 1ALGO)
    - Graphes (*cf.* cours 2GRAP de seconde année)
    - Arbres (*cf.* cours 2GRAP et 2ALGO de seconde année)
    - Etc.

1. Généralités sur les séquences.

Notion de séquence

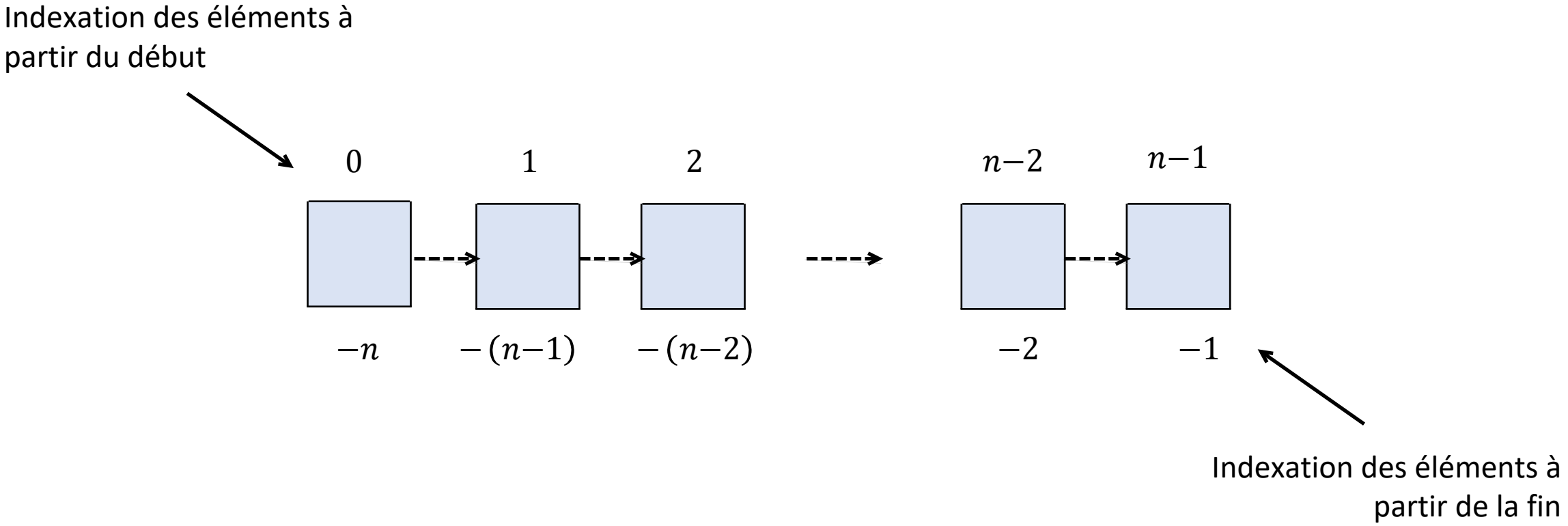
- Suite d’éléments accessibles par leur position. On parle également de rang ou d’indice.

- Chaque élément, à part le premier, a un prédécesseur et, à part le dernier, a un successeur.



1. Généralités sur les séquences.

Représentation imagée d'une séquence de n éléments



1. Généralités sur les séquences.

Les trois principaux types de séquences

1. Les listes dont les éléments sont quelconques et modifiables.

2. Les t-uples dont les éléments sont quelconques et non modifiables.

3. Les chaînes de caractères dont les éléments sont des caractères et ne sont pas modifiables.

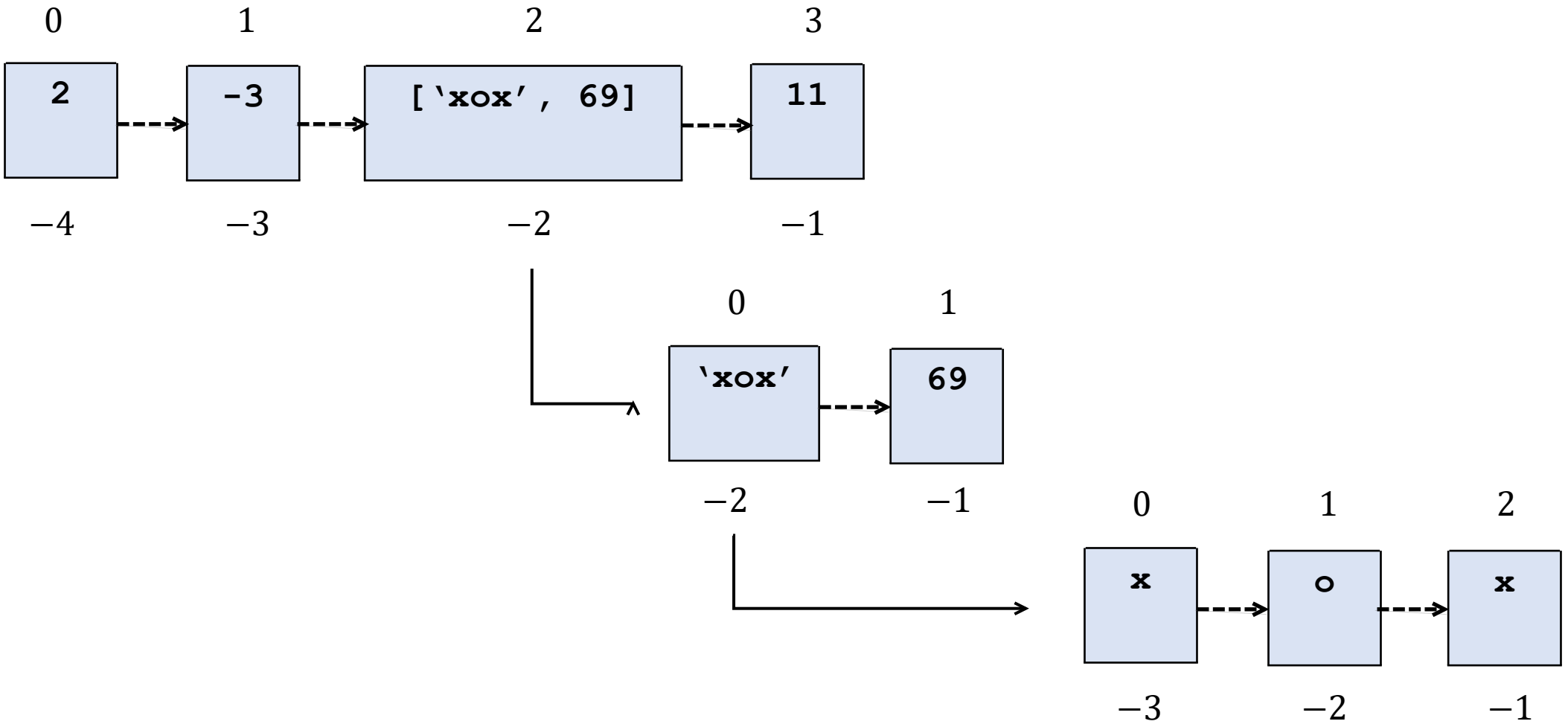
1. Généralités sur les séquences.

Principales syntaxes pour créer une séquence

```
myList = [item1, item2, …, itemN]

myList = list(otherStructure)

myTuple = (item1, item2, …, itemN)

myTuple = tuple(otherStructure)

myString = “text“
```

1. Généralités sur les séquences.

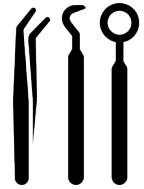
Exemple : représentation imagée de la liste [2, -3, ['xox', 69], 11]



1. Généralités sur les séquences.

Accès aux éléments d'une séquence

Opération	*Résultat*
s[i]	**i**-ème élément de **s**
s[i:j]	Sous-séquence de **s** constituée des éléments entre le **i**-ème (inclus) et le **j**-ème (exclus)
s[i:j:k]	Sous-séquence de **s** constituée des éléments entre le **i**-ème (inclus) et le **j**-ème (exclus) pris avec un pas de **k**



1. Généralités sur les séquences.

Opérations de traitement des séquences

Opération	*Résultat*
x in s	Teste si **x** appartient à **s**
x not in s	Teste si **x** n'appartient pas à **s**
s + t	Concaténation de **s** et **t**
s * n ou n * s	Concaténation de **n** copies de **s**
len(s)	Nombre d'éléments de **s**
min(s)	Plus petit élément de **s**
max(s)	Plus grand élément de **s**
s.count(x)	Nombre d'occurrences de **x** dans **s**
s.index(x)	Indice de **x** dans **s**.

1. Généralités sur les séquences.

Itération sur les éléments vs itération sur les indices

- Éléments :

```
for x in mySequence:
    traitement de l'élément x
```

- Indices :

```
for i in range(len(mySequence)):
    traitement de l'élément mySequence[i]
    et/ou de l'indice i
```

1. Généralités sur les séquences.

Itération sur les couples (indice, élément) et itération sur deux séquences

- Couples (indice, élément) :

```
for i, x in enumerate(mySequence):
    traitement de l'élément x
    et/ou de l'indice i
```

- Deux séquences :

```
for x, y in zip(mySequence1, mySequence2):
    traitement des éléments x et y
```

1. Généralités sur les séquences.

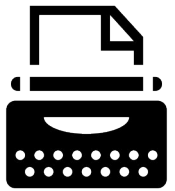
2. Les listes.

2. Les listes.

Deux conséquences de la muabilité des listes

1. On peut modifier les éléments d'une liste, en supprimer, en ajouter d'autres.

2. Une liste transmise en paramètre à un sous-programme peut être modifiée par cette dernière.



2. Les listes.

Exemple : modification d'une liste par un sous-programme

```
def swap(l, i, j):
    l[i], l[j] = l[j], l[i]

maListe = [1, 2, 3, 4, 5]
print("avant échange :",maListe)
swap(maListe,0,3)
print("après échange :",maListe)
```

```
avant échange : [1, 2, 3, 4, 5]
après échange : [4, 2, 3, 1, 5]
```

2. Les listes.

Opérations de traitement spécifiques aux listes

Opération	***Résultat***
s.append(x)	Ajoute l'élément **x** à la fin de **s**
s.extend(t)	Étend **s** avec la séquence **t**
s.insert(i,x)	Insère l'élément **x** à la position **i**
s.clear()	Supprime tous les éléments de **s**
s.remove(x)	Retire l'élément **x** de **s**
s.pop(i)	Renvoie l'élément d'indice **i** et le supprime
s.reverse()	Inverse l'ordre des éléments de **s**

2. Les listes.

Listes définies en compréhension : principe

- Le but est de définir une liste en effectuant des opérations sur chacun des éléments d'une séquence déjà existante.

```
[expression(x) for x in mySequence if condition]
```

2. Les listes.

Listes définies en compréhension : exemple

```
mySequence = tuple(range(0,10))
myList = [x**2 for x in mySequence if x%2==0]

print(myList)
```

```
[0, 4, 16, 36, 64]
```

2. Les listes.

Listes multidimensionnelles : principe

- Les éléments d'une liste peuvent eux-mêmes être des listes. On peut ainsi créer des listes multidimensionnelles.

- L'accès aux éléments se fait alors avec une syntaxe de la forme :

```
myList[i][j]
```

2. Les listes.

Listes multidimensionnelles : exemple

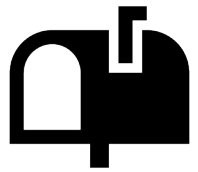
```
plateau = [[0]*3 for i in range(4)]
plateau[3][2] = 666
for i in range(4):
    for j in range(3):
        print(plateau[i][j], ' ', end='')
    print('\n')
```

```
0   0   0

0   0   0

0   0   0

0   0   666
```

2. Les listes.

Copie d’une liste : remarque importante

- La tentative de copie d’une liste avec la syntaxe naturelle « myList2 = myList1 » ne crée qu’un alias.
- Les deux noms pointeront vers le même emplacement mémoire.
- Il n’y aura donc en réalité qu’une seule liste existante et les modifications apportées à l’une seront répercutées sur l’autre.



2. Les listes.

Copie superficielle d'une liste : plusieurs syntaxes possibles

```
myList2 = myList1[:]

myList2 = list(myList1)

import copy
myList2 = copy.copy(myList1)

myList2 = [ ]
myList2.extend(myList1)
```

2. Les listes.

Copie profonde d'une liste : principe et syntaxe

- Dans le cas de listes imbriquées une copie superficielle ne suffit pas et il faut alors réaliser une copie profonde :

```
import copy
myList2 = copy.deepcopy(myList1)
```

2. Les listes.

3. Les chaînes de caractères.

3. Les chaînes de caractères.

Quelques opérations spécifiques aux caractères et chaînes de caractères

Opération	***Résultat***
ord(c)	Retourne l'entier représentant le code Unicode du caractère **c**
chr(n)	Retourne le caractère dont le code Unicode est représenté par l'entier **n**
s.lower()	Convertit la chaîne de caractères **s** en minuscules
s.upper()	Convertit la chaîne de caractères **s** en majuscules

3. Les chaînes de caractères.

Les premiers caractères de la table Unicode

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

3. Les chaînes de caractères.

Séparation et concaténation

- Séparation :

```
myText.split(sep)
```

- Concaténation :

```
sep.join(myListOfString)
```

3. Les chaînes de caractères.

4. Les t-uples.

4. Les t-uples.

Deux conséquences de l'immuabilité des t-uples

1. On ne peut pas modifier les éléments d'un t-uple, en supprimer, en ajouter d'autres.

2. Un t-uple transmis en paramètre à une fonction ne peut pas être modifié par cette dernière.

4. Les t-uples.

Exemple : non modification d'un t-uple par un sous-programme

```
def swap(l, i, j):
    l[i], l[j] = l[j], l[i]

monTuple = (1, 2, 3, 4, 5)
print("avant échange :", monTuple)
swap(monTuple,0,3)
print("après échange :", monTuple)
```

```
avant échange : (1, 2, 3, 4, 5)
Traceback (most recent call last): [...]
TypeError: 'tuple' object does not support item assignment
```

4. Les t-uples.

Quelques utilisations possibles de t-uples

- Si on veut définir une séquence de données que l'on ne souhaite pas modifier, *i.e.* une séquence constante, utiliser un t-uple sécurise ce fait.

- Itérer sur les éléments d'un t-uple est plus rapide que sur ceux d'une liste.

- Une fonction qui retourne "plusieurs valeurs", retourne en fait un t-uple.

- Les t-uples peuvent être utilisés comme des clés de dictionnaire (voir cours 1ALGO) ce qui n'est pas le cas des listes.

4. Les t-uples.

Exemple : utilisation d'un t-uple pour dénombrer un nombre de voyelles

```
def voyelles(mot):
    t = ('a', 'e', 'i', 'o', 'u')
    s = 0
    mot = mot.lower()
    for x in t:
        s += mot.count(x)
    return s

print(voyelles('Alamo'))
```

4. Les t-uples.