Les bases de la programmation informatique

avec Python

Deuxième partie : les types structurés

SOMMAIRE

I. CHAÎNES, LISTES, TUPLES ET DICTIONNAIRES

1. Présentation

- Les **chaines de caractères** ne sont pas modifiables (non mutable) :

```
>>> chaine='Roméo et Juliette'
```

- Les **listes** sont des sortes de tableaux pouvant contenir tous les types de données se trouvant entre crochets[], ils sont mutables :

liste =[' jambon ' , ' fromage ' , ' miel ' , ' confiture ' , ' chocolat ']

```
liste2=[1,2,3]

liste3=['toto',2,3.5,True,['b',liste2]]
```

- Les **tuples** ≈ listes non mutables, les données sont entre parenthèses ()

```
>>> tuple=('a','b','c','d')
```

- Les **dictionnaires**, mutables, portent bien leurs noms. Ils associent à un élément, un autre élément. Les éléments sont entre accolades

```
dicoAnglais={'computer': 'pc', 'mouse': 'souris', ' keyboard ': ' clavier '}
```

2. Eléments communs

Chaîne	Liste	Tuple	Dictionnaire
`chaine='Roméo et Juliette'`	`liste =[ ' jambon ' , ' fromage ' , ' miel ' , ' confiture ' , ' chocolat ']`	`tuple1=('a','b','c','d')`	`dicoAnglais={'computer': 'pc', 'mouse': 'souris', ' keyboard ': ' clavier '}`
Connaitre la taille len			
`>>> len (chaine)` `17`	`>>> len(liste)` `5`	`>>> len(tuple1)` `4`	`>>> len(dicoAnglais)` `3`
Test d'appartenance ou non appartenance avec in			
`>>> 'J' in chaine` `True`	`>>> 'jambon' in liste` `False`	`>>> 'jambon' in tuple1` `False`	`>>> 'jambon' in dicoAnglais` `False`
Concaténation avec+			
`>>> chaine+'3'` `'Roméo et Juliette3'`	`>>> liste + [3]` `[' jambon ', ' fromage ', ' miel ', ' confiture ', ' chocolat ', 3]`	`>>> tuple1+('3',) # attention à la virgule` `('a', 'b', 'c', 'd', '3')`	*`# pas pour les dictionnaires`*
Opération index pour trouver la première occurrence index			
`>>> chaine.index('J')` `9`	`>>> liste.index(' jambon ')` `0`	`>>> tuple1.index('b')` `1`	*`# pas pour les dictionnaires`*
Nombre de fois count			
`chaine.count('e')` `3`	`>>> liste.count('e')` `0`	`>>> tuple1.count('c')` `0`	*`# pas pour les dictionnaires`*
Min max (code ascii)			
`>>> max(chaine)` `'é'`	`>>> max(liste)` `' miel '`	`>>> min(tuple1)` `'a'`	`>>> max(dicoAnglais)` `'mouse'`
Copie n fois avec *			
`>>> chaine*2` `'Roméo et JulietteRoméo et Juliette'`	`>>> liste*2` `[' jambon ', ' fromage ', ' miel ', ' confiture ', ' chocolat ', ' jambon ', ' fromage ', ' miel ', ' confiture ', ' chocolat ']`	`>>> tuple1*2` `('a', 'b', 'c', 'd', 'a', 'b', 'c', 'd')`	*`# pas pour les dictionnaires`*
Trié avec sorted			
`>>> sorted(chaine)` `[' ', ' ', 'J', 'R', 'e', 'e', 'e', 'i', 'l', 'm', 'o', 'o', 't', 't', 't', 'u', 'é']`	`>>> sorted(liste)` `[' chocolat ', ' confiture ', ' fromage ', ' jambon ', ' miel ']`	`>>> sorted(tuple1)` `['a', 'b', 'c', 'd']`	`>>> sorted(dicoAnglais)` `[' keyboard ', 'computer', 'mouse']`
supprimer avec del			
`#suppression impossible`	`>>> del liste[2]`	*`#suppression impossible`*	*`#pas de cette façon`*

3. Le slicing : découpage

(d'après mooc FUN python 3)

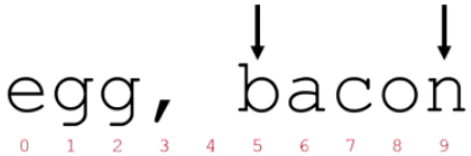
Le slice permet le découpage de structures de données, typiquement les chaînes de caractères ou les listes

a. Conventions de début et fin

- La borne de gauche est incluse et celle de droite est exclue :

```
>>> s[0:3] # de 0 inclus à 3 exclu
'egg'
```


```
>>> s[5:10]
'bacon'
```



b. Bornes omises

- Pas de borne de gauche : début jusqu'à la borne de droite -1

```
>>> s[:3]
'egg'
```

- Pas de borne de droite : à partir de la borne de gauche jusqu'à la fin

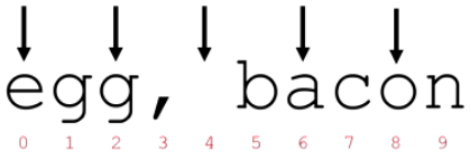
```
>>> s[5:]
'bacon'
```

- Sans borne : tous les éléments

```
>>> s[:]
'egg, bacon'
```

c. Avec un troisième paramètre : notion de pas

```
>>> s[0:10:2]
'eg ao'
```



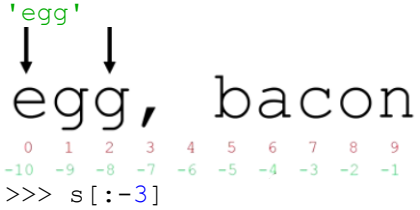
- Autres exemples :

```
>>> s [::2]# début à la fin par pas de 2
'eg ao'
>>> s[:8:3]# du début à 8 exclus par pas de 3
'e,a'
>>> s[2::3]#les éléments à partir de 2 jusqu'à la fin par pas de 3
'gbo'
```

d. Avec des indices négatifs

- Ils numérotés à partir de la fin, c'est un moyen commode d'accéder aux derniers éléments.

```
>>> s[-10:-7] # on parcourt toujours de gauche vers la droite
'egg'
```




```
>>> s[:-3]
'egg, ba'
```

- Parcourt de droite à gauche avec un pas de -1 :

```
>>> s[::]

'egg, bacon'

>>> s[::-1]

'nocab ,gge'
```

- Par contre pour afficher 'nocab' (bacon à l'envers)

```
>>> s[5:10:-1] # renvoie une chaîne vide
''
>>> s[9:5:-1]# il manque une lettre
'noca'
>>> s[9:4:-1]# car il ne faut pas oublier que la borne de droite n'est pas prise en
compte
'nocab'
```

4. Conversion entre chaîne<->liste

a. Chaine-> Liste avec split()

```
>>> s='un chaîne de caractères'
>>> liste=s.split()
>>> liste
['un', 'chaîne', 'de', 'caractères']
liste=chaine.split(':')
```

b. Liste ->Chaîne avec join()

```
>>> s=" ".join(liste)
>>> s
'un chaîne de caractères'
>>> s="-".join(liste)
>>> s
'un-chaîne-de-caractères'
```

II. LES CHAÎNES DE CARACTÈRES

1. Les chaînes sont comparables

```
    if mot < "limonade":
        place = "précède"
```

2. If utilisée avec in

```
car = "e"
voyelles = "aeiouyAEIOUYàâéèêëùîï"
if car in voyelles:
    print(car, "est une voyelle")
```

3. for avec les chaînes de caractères

```
nom='Cédric'
for car in nom:
    print(car+'*',end='')
```

4. Parcours d'une séquence : l'instruction for ... in ...

Exemple

```
nom='Cédric'
for car in nom:
    print(car+'*',end='')
```

5. Chaîne=objet

Voici quelques fonctions utilisables avec les chaînes :

- split() : découper
- join(liste) convertit liste en chaîne
- find(sch) : cherche la position d ' une sous-chaîne sch dans la chaîne
- count(sch) : compte le nombre de sous-chaînes sch
- Upper(), lower() et capitalize()
- lower() : convertit une chaîne en minuscules :
- upper() : convertit une chaîne en majuscules :
- title() : convertit en majuscule l ' initiale de chaque mot (suivant l'usage des titres anglais) :
- capitalize() : variante de la précédente. Convertit en majuscule seulement la première lettre de la chaîne :
- swapcase() : convertit toutes les majuscules en minuscules, et vice-versa :
- strip() : enlève les espaces éventuels au début et à la fin
- replace(c1, c2) : remplace tous les caractères c1 par c2
- index(car) : retrouve l ' indice (index) de la première occurrence du caractère

Exemples :

```
>>> x="HellO"

>>> x.upper()

'HELLO'

>>> ch = "Le Lièvre Et La Tortue"

>>> print(ch.swapcase())
```

lE lIÈVRE eT lA tORTUE

```
>>> ch8 = "Si ce n'est toi c'est donc ton frère"

>>> print(ch8.replace(" ","*"))

Si*ce*n ' est*toi*c ' est*donc*ton*frèr
```

6. Retour sur la fontion print()

- + : concaténation
- \n : permet de passer une ligne.
- * n : répétition

```
>>> print("Bonjour", "à", "tous", sep ="*")
Bonjour*à*tous
>>> print("Bonjour", "à", "tous", sep ="")
Bonjouràtous

>>> print(1, 'a', val,end =' ');print(" sur la même ligne")
1 a 3.5  sur la même ligne
```

7. Afficher des nombres

Code Python	Affichage
`for number in range(4):` `    print(' ' + str(number))`	vec pass Number is 0 Avec pass Number is 1

	Avec pass Number is 2 Avec pass Number is 3

8. Formatage

Elle peut se faire avec .format ou les f-string si version de python est 3.6

a. Version la plus simple

```
>>> nombre=10
>>> multiplicateur=3
>>> print( nombre,"*",multiplicateur,"=",nombre*multiplicateur)
10 * 3 = 30
```

Remarquez les espaces

Elle peut se faire avec .format ou les f-string si version de python est 3.6

```
>>> print( f'{nombre} * {multiplicateur} = {nombre*multiplicateur}')
10 * 3 = 30
```

```
>>> print( '{} * {} = {}'.format(nombre,multiplicateur,nombre*multiplicateur))
10 * 3 = 30
```

```
>>> print( nombre,' * ',multiplicateur, ' = ' ,nombre*multiplicateur))
10 * 3 = 30
```

III. LES LISTES (MUTABLE)

Pour stocker des données mutables, c'est une séquence comme les chaînes.
Elles ne stockent par réellement les données, mais des références vers ces objets et par conséquent la taille des objets listes est indépendant les objets les contenant, donc extrêmement efficace au niveau mémoire.

1. Différents types

```
On peut aussi mélanger des objets de types différents dans la même liste :
[1, 'a', 2, 3, 3.14159, True, False, 1]
On peut même avoir une liste qui contient une liste :
[1, [2, 3], 'toto']
```

Une liste peut être vide :

```
[]
```

Les éléments d'une liste peuvent contenir des expressions complexes, faisant appel à des variables ou des fonctions :

```
a = 2
[(2 + 3) * a, a * 2]
```

2. Ajouter un éléments #concaténer un élément

```
>>> x = ["a", "b", "c"]
>>> x=x+["d"] #concaténer un élément idem x.append("d")
>>> x
['a', 'b', 'c', 'd']
```

3. Supprimer un ou des éléments par del() ou pop()

```
>>> del(x[1])
>>> x
['a', 'c', 'd']
```

Pop() supprime et renvoie l'élément

```
>>> x.pop(2)
'c'
>>> x
['a', 'b', 'g', 'h']
```

4. Avec les slices

```
>>> L=['spam','Spam','SPAM!']
>>> L[1]='eggs'           #assignation par index
>>> L
['spam', 'eggs', 'SPAM!']
>>> L[0:2]=['eat','more']  #suppression et insertion
>>> L
['eat', 'more', 'SPAM!']
>>> L[1:1]=['eggs']        #uniquement insertion
>>> L
['eat', 'eggs', 'more', 'SPAM!']
>>> L[0:2]=[]         #suppression
>>> L
['more', 'SPAM!']
>>> L[1:1]=['eat','eggs']     #uniquement insertion de plusieurs éléments
>>> L
['more', 'eat', 'eggs', 'SPAM!']
>>>
```

5. Suppression / remplacement d'éléments

```
>>> mots

['jambon','fromage','miel','confiture','chocolat','saucisson','ketchup']

>>> mots[2:5] = [] # [] désigne une liste vide

>>> mots

['jambon','fromage','saucisson','ketchup']

>>> mots[1:3] = ['salade'] # suppression et insertion

>>> mots

['jambon','salade','ketchup']

>>> mots[1:] = ['mayonnaise','poulet','tomate'] # suppression jusqu'à la fin et insertion

>>> mots

['jambon','mayonnaise','poulet','tomate']
```

À la première ligne de cet exemple, nous remplaçons la tranche [2:5] par une liste vide, ce qui correspond à un effacement.
À la quatrième ligne, nous remplaçons une tranche par un seul élément. Notez encore une fois que cet élément doit lui-même être « présenté » comme une liste.
À la 7 e ligne, nous remplaçons une tranche de deux éléments par une autre qui en comporte 3.

6. Append(liste) pour ajouter un élément

Attention : avec append on ajoute une liste pas les éléments

```
>>> x.append(y)
>>> x
['a', 'b', 'g', 'h', 'aa', 'bb', 'aa', 'bb', ['aa', 'bb', [...]]]
```

7. Création d'une liste par compréhension

On peut aussi définir une liste par compréhension avec la syntaxe :

```
[ fonction(x) for x in liste if test(x) ]
```

a. de nombres à l'aide de la fonction range()

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

b. Création avec [__ for i in range____]

```
>>> liste=[i for i in range(0,20,2)]
>>> liste
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

c. D'autres exemples :

```
>>> [2*x for x in range(10)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18
>>> [x for x in range(10) if x>5
[6, 7, 8, 9
>>> [2*x for x in range(10) if x>5
[12, 14, 16, 18
>>> [2*x for x in range(10) if 2*x>5
[6, 8, 10, 12, 14, 16, 18
>>> [x for x in range(10) if x>5 and x<8
[6, 7
>>> [x+1 for x in [2*x for x in range(10)
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19
>>> [2*x for x in [x+1 for x in range(10)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20
>>> [i*2 for i in range(10)
[0, 2, 4, 6, 8, 10 ,12 ,14 ,16 ,18 ,20
>>> [i for i in range(10) if i % 2 == 0
[0, 2, 4, 6, 8]
```

Un peu plus compliqué : tous les nombres premiers <=50 :

8. For

a. Exemple

```
liste=['Cédric','Loïc','Joelle','Pierre']
for nom in liste:
    print('La longueur de '+nom+' est '+str(len(nom)))#mettre des espaces
```

L'affichage est :
La longueur de Cédric est 6
La longueur de Loïc est 4
La longueur de Joelle est 6
La longueur de Pierre est 6

9. Résumé

Operation	Interpretation
LI = []	#An empty list
L2 = [0, 1, 2, 3]	#Four items: indexes 0..3
L3 = ['abc', ['def', 'ghi']]	#Nested sublists

`L2[i]`	`#Index, index of index, slice, length`
`L3[i][j]`	
`L2[i:j]`	
`len(L2)`	
`Li + L2`	`#Concatenate, repeat`
`L2 * 3`	
`for x in L2`	`#Iteration, membership`
`3 in L2`	
`L2.append(4)`	`#Methods: grow, sort, search, insert, reverse, l2.pop() l2.pop(4)`
`L2.extend([5,6,7])`	
`L2.index(1)`	
`L2.reverse()`	
`L2.sort()`	`L3=sorted(L2)#renvoie un obj contenant l2 trié`
`L2.insert(2, 8)`	`Idem l2[2]=8`
`del L2[k]`	
`del L2[i:j]`	`L2.pop()#renvoie le dernier`
`L2.remove(2)`	`L2[i:j] = []`
`L2[i] = 1`	`#idem l2.insert(i,1)`
`L2[2:5] = [4,5,6]`	`#remplacements`
`L2[2:2] = [4,5,6]`	`#insertion à l'emplacement 2`
`ll=list(range(4))`	`#Make lists/tuples of integers`
`ll=list(range(0, 4))`	
`L4 = [x**2 for x in range(5)]`	`#List comprehensions`

`>>> min(nonPremiers)`
 	
`4`
 	
`>>> max(nonPremiers)`
 	
`49`	

IV. LES TUPLES (NON MUTABLE MAIS MÉLANGE)

1. Opérations courantes

`>>> t=(0,)`
`>>> t2=(0)`
`>>> t= True,3.4,18,'ee'`
 	
`>>> type(t)`
 	
`<class 'tuple'>`
 	
`>>> t=(0)`
 	
`>>> type(t)`
 	
`<class 'int'>`
	A one-item tuple
	Attention à la virgule
	Les parathèses sont factultatives

`t=(0,'no',1.2,3)`	Un tuple à 4 éléments

`>>> T=1,2,3,4`	Création d'un tuple mais sans parenthèse
`>>> t1=('rtk',('bob',3))`	Un tuple imbriqué

`>>> t1[0]`
`'rtk'`
`>>> t1[1][1]`
`3`
	Indexé

`t2 = tuple('spam')`
`print(t2)`
`('s', 'p', 'a', 'm')`
	Convertit en tuple

`>>> t+t2`
`(0, 'no', 1.2, 3, 's', 'p', 'a', 'm')`
`>>> t*3`
`(0, 'no', 1.2, 3, 0, 'no', 1.2, 3, 0, 'no', 1.2, 3)`
	Concaténation et répétition

`>>> for x in t : print(x,end=',')` `0,no,1.2,3,`	Itération
`>>> 'no' in t` `True`	Appartenance

2. Convertion entre tuple et liste

```
>>> t=('s', 'p', 'a', 'm')
>>> liste=list(t)
>>> liste
['s', 'p', 'a', 'm']
>>> liste.sort()
>>> liste
['a', 'm', 'p', 's']
>>> T=tuple(liste)
>>> T
('a', 'm', 'p', 's')
```

V. LES DICTIONNAIRES – MUTABLES -

Il est utilisé pour stocker des pairs de clefs/valeurs et ne sont pas trié.

1. Création

a. Simple

```
>>> dico={}
>>> dico['computer']='pc'
>>> dico
{'computer': 'pc'}
>>> dico['mouse']='souris'
>>> dico['computer']
'pc'
```

Pour ajouter : par de append() mais directement une paire de clé-valeur.

b. Avec des dictionnaires de dictionnaire

```
>>> ang2fr = {'one':'un', 'two': 'deux', 'three': 'trois'}
>>> type(ang2fr)
<class 'dict'>
>>> d = {'ent': 1, 'list': [1, 2, 3], 1:'un',  (1,2): 3.14, 'dic': ang2fr}
>>> d[[3,4]] = 45
```

2. Création avec [__ for i in range____]

```
>>> ascii={i:chr(i) for i in range(65,90)}
>>> ascii
{65: 'A', 66: 'B', 67: 'C', 68: 'D', 69: 'E', 70: 'F', 71: 'G', 72: 'H', 73: 'I',
```

3. Accès par la clef ou par slice

```
>>> dico['computer']
'pc'
```

```
>>> d['list'][-1]
3
```

4. Opérations sur les dictionnaires :

```
>>> del dico['computer']
>>> dico
{'mouse': 'souris', ' keyboard ': ' clavier '}
```

```
>>> ascii.pop(66)

'B'
```

```
>>> ang2fr
{'one': 'un', 'two': 'deux', 'three': 'trois', 'four': 'quatre'}
>>> len(ang2fr)
4
>>> 'one' in ang2fr
True
>>> 'deux' in ang2fr
False
```

5. Méthodes

a. Les clés et les valeurs keys() et values()

```
>>> dico.keys()
dict_keys(['mouse',' keyboard'])
>>> dico.values()
dict_values(['souris',' clavier'])
print(dico.items())
>>>dict_items([('computer', 'pc'), ('mouse', 'souris')])
```

a) Parcourir toutes les clefs

```
inventaire = {'pommes': 430,'bananes': 312,'oranges': 274,'poires': 137}
for clef in inventaire.keys():
    print('clé',clef,'valeur',inventaire[x])
```

Affichage :

```
clé pommes valeur 430
```

b) Clés et valeurs

```
print(inventaire.items())
#dict_items([('pommes', 430), ('bananes', 312), ('oranges', 274), ('poires', 137)])
print(list(inventaire.items()))
#[('pommes', 430), ('bananes', 312), ('oranges', 274), ('poires', 137)] par intéressant
print(tuple(inventaire.items()))
#(('pommes', 430), ('bananes', 312), ('oranges', 274), ('poires', 137))
```

b. Recopie des données = ? mauvaise idée

```
>>> inventaire
{'pommes ': 430, 'bananes ': 312, 'oranges ': 274, 'poires ': 137}
```

```
>>> id(inventaire)
53900056
>>> stock= inventaire
>>> id(inventaire)
53900056
```

Les deux variables pointent sur le même emplacement mémoire donc

```
>>> del inventaire ['bananes ']
>>> stock
{'pommes ': 430, 'oranges ': 274, 'poires ': 137}
```

c. Avec copie c'est mieux

```
>>> stock= inventaire.copy()
>>> inventaire ['raisin']=430
>>> stock
{'pommes ': 430, 'oranges ': 274, 'poires ': 137}
>>> inventaire
{'pommes ': 430, 'oranges ': 274, 'poires ': 137, 'raisin': 430}
```

d. Suppression avec pop()

Suppression + renvoie la valeur

```
>>> stock={'pommes': 430, 'oranges': 274, 'poires': 137}
>>> stock.pop('oranges')
274
>>> stock
{'pommes': 430, 'poires': 137}
```

e. Supprimer tout le dictionnaire

a) Avec l'utilisation d'une liste

Impossible avec pop :

```
>>> for i in stock : stock.pop(i)
430
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in <module>
    for i in stock : stock.pop(i)
RuntimeError: dictionary changed size during iteration
```

L'idée est de créer une liste d'éléments à supprimer :

```
stock={'pommes': 430, 'oranges': 274, 'poires': 137}
to_remove=[]
for i in stock:# if stock[i]<400: sous condition
    to_remove.append(i)
for i in to_remove:
    stock.pop(i)
print(to_remove,stock)
```

L'affichage est

```
['pommes', 'oranges', 'poires'] {}
```

6. Parcourir un dictionnaire

```
>>> for clef,valeur in stock.items():
   print(clef,valeur)
#affichage
 pommes  430
 oranges  274
 poires  137
```

7. Les clés ne sont pas nécessairement des chaînes de caractères

```
arbre[(1,2)] ='Peuplier'
arbre[(3,4)] ='Platane'
```

```
>>> dict = {('Ghostbusters',2016): 5.4,
('Ghostbusters',1984):7.8}
```

8. Accès aux données par la méthode get()

a. Sans get

```
>>> arbre[(1,3)]
Traceback (most recent call last):
 File "<pyshell#2>", line 1, in <module>
    arbre[(1,3)]
KeyError: (1, 3)
```

b. get

Pour palier cela : .get()

```
>>> arbre.get((6,5))
'Palmier'
>>> arbre.get((1,3))
>>> print(arbre.get((1,3)))
None
```

c. Attention à arbre[2]='salami'

Il ne s'agit pas de mettre à l'emplacement la valeur 'salami' mais d'ajouter 'salami' avec une clef de 2 :

```
>>> arbre[2]='salami'
>>> arbre
{(1, 2): 'Peuplier', (3, 4): 'Platane', (6, 5): 'Palmier', (5, 1): 'Cycas', (7, 3):
'Sapin', 2: 'salami'}
```

9. Construire des dictionnaires avec zip

a. Sans for et avec dict()

```
>>> clefs = ['spam', 'eggs', 'toast']
>>> valeurs = [1,3,5]
>>> d3=dict(zip(clefs,valeurs))
>>> d3
{'spam': 1, 'eggs': 3, 'toast': 5}
```

10. Remarques

a. La clef doit être non mutable

Par exemple une clé de type *list* n'est pas accepté !

```
TypeError: unhashable type: 'list'.
```

Les non mutables sont int, float, bool, str et tuples uniquement.

L'idée est que Python ne désire pas gérer des ensembles dont les **valeurs des éléments** pourraient changer en cours de route ; de même pour des clés de dictionnaires potentiellement modifiées durant l'exécution du programme.

b. Avec for

Python impose une restriction à l'utilisation de l'instruction for pour itérer tous les éléments d'un dictionnaire ou d'un ensemble : le corps du for ne peut ajouter ou supprimer de composante dans la séquence sur laquelle il itère. Par exemple le code suivant provoque une erreur à l'exécution :

```
s = {'a', 'b', 'c', 'd', 'e'}
for i in s:
    if i in 'aeiouy':
        s.remove(i)

Traceback (most recent call last):
  File "<input>", line 1, in <module>
RuntimeError: Set changed size during iteration
```