Programmation 2

Slides modified from **https://github.com/UGE-IGM/courspython**

Chapitre 6 : Itérables - Mutables

Les itérables

La doc python indique:

> *"un itérable est une structure de données permettant d'accèder à ses éléments un par un"*

Par exemple, les `list` et les `str` sont des *itérables*. Tous les *itérables* permettent d'utiliser l'opérateur `in`. Pour un itérable `L`, on peut se servir de cet opérateur comme suit :

- `x in L`, `x not in L` : test d'appartenance
- `for elem in L:` parcours

Le deuxième cas d'utilisation permet de passer d'un élément de l'itérable au suivant. Vous avez déjà vu un cas particulier de cette syntaxe: `for i in range(n)`. En effet, `range(n)` est aussi un itérable (représentant la séquence 0,1,...,n-1).

Afficher chaque élément d'une liste:

In [7]:

```
l=[5,'toc',1.4,18,42,"tic"]
```

In [ ]:

```
def element(l):
    for elem in l:
       print(elem)
```

Les listes sont *mutables* c'est à dire que l'on peut changer leur contenu (au lieu de créer une nouvelle liste).

In [8]:

```
print(l)
l[2]=111
print(l)
```

```
[5, 'toc', 1.4, 18, 42, 'tic']
[5, 'toc', 111, 18, 42, 'tic']
```

Intervalles d'entiers : `range`

La fonction `range` fabrique des intervalles d'entiers:

- `range(i)` : entiers de `0` à `i-1`
- `range(i, j)` : entiers de `i` à `j-1`
- `range(i, j, k)` : entiers de `i` à `j-1` par pas de `k`

Len opérations autorisées sur un `range r` sont :

- `x in r, x not in r`
- `r[i]`
- `len(r), min(r), max(r), r.index(x), r.count(x)`
- `list(r)` (conversion en liste)

In [1]:

```
r = range(10)
print(r)
```

```
range(0, 10)
```

In [5]:

```
r[0]
```

Out[5]:

```
0
```

In [6]:

```
r[1]
```

Out[6]:

```
1
```

In [7]:

```
max(r)
```

Out[7]:

```
9
```

In [8]:

```
r = range(10, 30, 7)
r.index(17)
```

Out[8]:

```
1
```

In [10]:

```
len(r)
```

Out[10]:

```
3
```

In [11]:

```
list(r)
```

Out[11]:

```
[10, 17, 24]
```

In [12]:

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

Toutes les autres opérations sont interdites sur un range (car un range est non mutable). Plus précisément, sont interdits :

```
Concaténation, répétition, affectation d'élément
Autres méthodes de listes (append, etc.)
```

Caractéristique importante :
Un `range` n'est pas construit entièrement en mémoire. Au lieu de cela, ses éléments sont fabriqués « à la demande ». Donc, par exemple, `range(10000)` ne prend pas plus de place que `range(2)`.
Utilisation fréquente : parcours des indices d'une liste

In [14]:

```
lst = ['a', 'b', 'c']
for i in range(len(lst)):
    print(i, '->', lst[i])
```

```
0 -> a
1 -> b
2 -> c
```

Exercice : Écrire une fonction `chercher_range(iterable, x)` qui recherche la première position de `x` dans `iterable` en utilisant une boucle `for` et la fonction `range`.

In [18]:

```
def chercher_range(iterable, x):
    for i in range(len(iterable)):
        if lst[i] == x:
            return i
    return None
```

Un dernier type de parcours de liste : `enumerate`

La connaissance de cette notion n'est pas exigible à l'examen.
La fonction `enumerate(iterable)` permet d'itérer sur les couples `(i, elem)` constitués d'un indice et de l'élément correspondant dans `iterable`. Pour reprendre l'exemple précédent :

In [5]:

```
lst = ['a', 'b', 'c']
for i, elem in enumerate(lst):
    print(i, '->', elem)
```

```
0 -> a
1 -> b
2 -> c
```

In [2]:

```
def chercher_enumerate(L, x):
    for i, elem in enumerate(L):
        if elem == x:
            return i
    return None
```

Retour sur les chaînes de caractères (type `str`)

Les chaînes de caractères (`str`) sont aussi des itérables !
Type `str` : **itérable** dont les éléments sont des caractères.

Les `str` sont **non mutables** c'est à dire qu'on ne peut pas changer leur contenu. Par exemple, ils interdisent l'affectation (`s[i] = x`), ainsi que `s.append('a')`, `s1.extend(s2)`, `s.insert(i, x)`, `s.pop()`, `s.pop(i)`, `s.remove(x)`, `s.clear()`, `s.reverse()`,...

En tant qu'itérables, les chaînes autorisent :

- `s[i]` : accès à un caractère par son indice
- `len(s)` : longueur
- `min(s)`, `max(s)` : minimum et maximum
- `s1 in s2`, `s1 not in s2` : recherche de sous-chaîne
- `s2.index(s1)` : position de sous-chaîne
- `s2.count(s1)` : comptage de sous-chaînes
- `s1 + s2`, `s * 3` : concaténation et répétition

En tant qu'objets **itérables**, les chaînes peuvent être utilisées dans des boucles `for` :

In [6]:

```
s = 'Hildegarde'
# afficher les caractères de s, un par ligne
for c in s:
    print(c)
```

```
H
i
l
d
e
g
a
r
d
e
```

... et acceptent la conversion en liste :

In [22]:

```
s = 'Hildegarde'
list(s)
```

Out[22]:

```
['H', 'i', 'l', 'd', 'e', 'g', 'a', 'r', 'd', 'e']
```

Méthodes sur les chaînes

De très nombreuses méthodes existent sur les chaînes qui renvoient toutes une nouvelle chaîne:

In [8]:

```
s='ceci est un titre'
print(s.capitalize()) # renvoie une nouvelle chaine
print(s) # str est non mutable, s n'a donc pas été modifiée
```

```
Ceci est un titre
ceci est un titre
```

In [31]:

```
print(s.find("st"))
```

```
6
```

In [14]:

```
" ".join(["les", "aventures", "de", "python"])
```

Out[14]:

```
'les aventures de python'
```

In [15]:

```
" # ".join(["les", "aventures", "de", "python"])
```

Out[15]:

```
'les # aventures # de # python'
```

In [17]:

```
print(s.split(' '))
```

```
['ceci', 'est', 'un', 'titre']
```

In [18]:

```
print(s.split('e'))
```

```
['c', 'ci ', 'st un titr', '']
```

Mutables et non mutables

Les listes sont des structures *mutables* c'est à dire qu'on peut changer leur contenu (sans créer une nouvelle liste).

In [19]:

```
l = [5,4,3,2,1]
l[0] = 9
print(l)
```

```
[9, 4, 3, 2, 1]
```

Voir python tutor : c'est `l[0]` qui représente une nouvelle valeur (ici 9), `l` n'a pas bougé !
Par contre les chaines (comme les `int`) ne sont pas mutables.

In [20]:

```
s = 'una chaine'
s[2] = 'e'
```

```
---------------------------------------------------------------------
-------------
TypeError                                 Traceback (most rece
nt call last)
<ipython-input-20-7dcb6f2f7dbb> in <module>()
      1 s = 'una chaine'
----> 2 s[2] = 'e'

TypeError: 'str' object does not support item assignment
```

Toutes les opérations sur les str créent de nouvelles chaines.
Essayez avec python tutor le code suivant :

In [23]:

```
m = "une"
m = m + " chaine"
print(m)
```

```
une chaine
```

On a créé une nouvelle chaine 'une chaine' et déplacé l'étiquette m

Tuples

Ils sont similaires aux listes mais il sont **non mutables**.
Ils s'utilisent par exemple pour mémoriser les coordonnées d'un point fixe.
On les note avec des parenthèses:

In [24]:

```
couple = (2,64)
print(couple)
type(couple)
```

```
(2, 64)
```

Out[24]:

```
tuple
```

In [26]:

```
print(couple[1])
couple[1] = 42 # erreur non- mutable
```

```
64

---------------------------------------------------------------------
-------------
TypeError                                 Traceback (most rece
nt call last)
<ipython-input-26-42c11f5bc1d9> in <module>()
      1 print(couple[1])
----> 2 couple[1] = 42 # erreur non- mutable

TypeError: 'tuple' object does not support item assignment
```

Fichiers

Il est très facile de consulter le contenu d'un fichier texte en Python, et de le modifier. On accède la plupart du temps aux lignes successives d'un fichier à l'aide d'une boucle `for`.

Descripteurs de fichiers (type `file`)

Ouverture d'un fichier : `f = open(chemin, mode)`

- Paramètre `chemin` (chaîne de caractère) : nom (absolu ou relatif) d'un fichier
- Paramètre `mode` (chaîne de caractère) :
    - lecture seule : `'r'` (par défaut)
    - écriture (crée / écrase le fichier) : `'w'`
    - ajout (écrit à la fin) : `'a'`
    - *autres modes possibles pour fichiers binaires*

Fermeture : `f.close()`

Lire et écrire dans un fichier

Lecture (mode `'r'`) :

- Tout lire d'un coup (une seule chaîne) : `contenu = f.read()`
  Attention : charge tout le fichier en mémoire !
  Attention : inclut les retours à la ligne

- Liste des lignes (liste de chaînes) : `lst_lignes = f.readlines()`
  Attention : charge tout le fichier en mémoire !
  Attention : inclut les retours à la ligne
- Prochaine ligne : `ligne = f.readline()`
  Attention : renvoie une chaîne vide à la fin
- Parcours ligne par ligne : `for ligne in f: ...`

Écriture (modes `'w'` et `'a'`) : `f.write(chaine)`
Attention : n'inclut pas de retour à la ligne (`'\n'`) automatique !

In [ ]:

```
f = open('test.txt', 'w')
for chaine in ['bonjour', 'tout', 'le', 'monde !']:
    f.write(chaine + '\n')
f.close()

f = open('test.txt', 'a')
f.write('(signé : Machin)\n')
f.close()

f = open('test.txt', 'r')
for ligne in f:
    # strip supprime les espaces avant et après la chaîne
    print(ligne.strip())
f.close()
```

In [ ]:

```
f = open('8-dictionnaires-fichiers.ipynb')
for ligne in f:
    print(ligne.strip())
f.close()
```