Chaînes de caractères

1er novembre 2015

1 Chaînes et type `str` en `Python`

1.1 Définition

Pour représenter des textes, on utilise la structure de données de « chaîne de caractères » (*string* en anglais). En `Python`, cela correspond aux objets de type `str`.

Ces objets sont non-mutables, c'est-à-dire qu'une fois créés, on ne peut pas les modifier. Comme il ne peut y avoir des problèmes d'alias (par exemple, comme pour les listes `Python`), nous ne détaillerons pas ici leur représentation en mémoire.

1.2 Création et lecture

On définit une chaîne de caractère en entourant ces caractères par des guillemets simples, doubles, ou trois guillemets simples ou doubles. L'utilisation de guillemets simples permet d'utiliser des guillemets doubles dans la chaîne de caractère, et vice-versa.

```
>>> 'Hallo Welt'
'Hallo Welt'
>>> 'It is "Hello World" !'
'It is "Hello World" !'
>>> "Ja, aber 'not in germany' ..."
"Ja, aber 'not in germany' ..."
```

On peut signaler une tabulation par `\t` et créer une nouvelle ligne par `\n`.

```
>>> s = "\tL'albatros\nSouvent, pour s'amuser, les hommes d'équipage\n[...]"
>>> print(s)
    L'albatros
Souvent, pour s'amuser, les hommes d'équipage
[...]
```

On accède alors aux caractères d'une chaîne en donnant son indice, comme pour les listes.

```
>>> s[0]
'\t'
>>> s[-1]
']'
```

Remarquons que le tranchage fonctionne similairement aux listes.

```
>>> s[12:19]
'Souvent'
>>> s[:11]
"\tL'albatros"
>>> s[36:]
"les hommes d'équipage\n[...]"
```

On peut aussi créer une chaîne de caractères à partir de nombres ou de booléens avec la fonction `str`.

```
>>> str(42)
'42'
>>> str(-3.5)
'-3.5'
>>> str(4==4.)
'True'
```

Enfin, rappelons que ce type est non mutable.

```
>>> s[0] = "a"
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    s[0] = "a"
TypeError: 'str' object does not support item assignment
```

1.3 Opérations sur les chaînes

Comme pour les listes, on peut concaténer deux chaînes de caractères avec l'opérateur `+`.

```
>>> 'Hello'+'World'
'HelloWorld'
```

On peut aussi dupliquer une chaîne de caractères plusieurs fois avec l'opérateur `*`.

```
>>> print('Noober : What about now ?\n'*5)
Noober : What about now ?
Noober : What about now ?
Noober : What about now ?
Noober : What about now ?
Noober : What about now ?
```

La fonction `len` permet là encore de calculer la longueur d'une liste.

```
>>> len(s)
63
```

On peut aussi chercher la présence d'un motif dans une chaîne avec `in`, son absence avec `not in`

```
>>> 'Albatros' in s
False
>>> 'albatros' not in s
False
```

1.4 Chaînes et méthodes

De nombreuses méthodes existent sur les chaînes, voici les plus utiles.

Méthode	Description
`count(m)`	Compte le nombre d'occurences du motif `m`, sans chevauchement.
`find(m)`	Renvoie la première occurence du motif `m`.
`islower() (et autres)`	Renvoie un booléen indiquant si la chaîne est en minuscules.
`join(bout)`	Concatène les éléments de la liste `Python` `bout`, séparés par la chaîne.
`lower()`	Met en minuscule la chaîne de caractères.
`split(sep)`	Sépare la chaîne selon le séparateur `sep`
`strip(char)`	Enlève les lettres en début/fin si elles sont dans la chaîne `char`.
`upper()`	Mets en majuscules la chaîne.

FIGURE 1 – Quelques méthodes sur les chaînes de caractères.

```
>>> s = "    que ; d'espaces ; mes ; amis ! ;   "
>>> s.count('es')
3
>>> s.find('es')
12
>>> s.islower()
True
```

```
>>> '+'.join([str(i) for i in range(14)])
'0+1+2+3+4+5+6+7+8+9+10+11+12+13'
>>> 'HAHAHAHAHA !'.lower()
'hahahahaha !'
>>> s.split(';')
['    que ', " d'espaces ", ' mes ', ' amis ! ', '    ']
>>> s.split()
['que', ';', "d'espaces", ';', 'mes', ';', 'amis', '!', ';']
>>> s.strip()
"que ; d'espaces ; mes ; amis ! ;"
>>> s.strip('q; ')
"ue ; d'espaces ; mes ; amis !"
>>> s.upper()
"    QUE ; D'ESPACES ; MES ; AMIS ! ;   "
```

2 Algorithme : recherche d'un mot dans un texte

Le seul algorithme sur les chaînes que vous devez pouvoir reproduire est celui de la recherche d'un mot `m` dans une chaîne de caractères `s`.

Nous allons écrire un algorithme naïf. On commence par calculer la longueur de `m`. Ensuite, on parcout séquentiellement la chaîne `s` en regardant à chaque fois si le mot se trouve au début de l'emplacement considéré.

Cela donne l'écriture suivante.

```
def recherche(m,s):
    """Recherche le mot m dans la chaîne s
       Préconditions : m et s sont des chaînes de caractères"""
    long_s = len(s) # Longueur de s
    long_m = len(m) # Longueur de m
    for i in range(long_s-long_m):
        # Invariant : m n'a pas été trouvé dans s[0:i+long_m-1]
        if s[i:i+long_m] == m: # On a trouvé m
            return True
    return False

print(recherche('42','Les 7 samouraïs'))
print(recherche('7','Les 7 samouraïs'))
```

Cela renvoie alors :

```
False
True
```

Remarquons que l'on aurait pu exécuter cet algorithme sur une liste `Python`...

3 Exercices

Exercice 3.0.1. Écrire une fonction `c_recherche(m,s)` de recherche indiquant si le mot `m` est dans la chaîne `s` insensible à la casse (majuscules ou minuscules).

Exercice 3.0.2. Écrire une fonction `i_recherche(m,s)` de recherche du mot `m` dans une chaîne `s`, renvoyant la position de la première occurence du mot dans la chaîne et une erreur si le mot n'est pas dans le texte.

Exercice 3.0.3. Écrire une fonction `compte(m,s)` comptant le nombre d'occurences du mot `m` dans une chaîne `s`, chevauchements compris.

Exercice 3.0.4. Écrire une fonction `dist_compte(m,s)` comptant le nombre d'occurences du mot `m` dans une chaîne `s`, sans chevauchements.