Informatique IX
Chaînes de caractères

Laurent Kaczmarek

PCSI^2 2013-2014
Lycée Louis Le Grand

Lundi 18 novembre 2013

Définir une chaîne

- En plus des nombres Python peut aussi manipuler des chaînes, qui peuvent être exprimées de différentes façons. Le type est `str`. Les chaînes peuvent être délimitées par des simples quotes (apostrophes) ou doubles quotes (guillemets).

- Exemples :

  - ```
    >>> a='bou'
    >>> a
    'bou'
    >>> print(a)
    bou
    ```

  - ```
    >>> c='"Oui",
    dis-je.'
    >>> print(c)
    "Oui", dis-je.
    ```
  - ```
    >>> d="L'art"
    >>> print(d)
    L'art
    ```
    ```

Définir une chaîne

- La chaîne vide est notée ' '.

Les caractères accentués

- Nous notons que les chaînes admettent ou non les caractères accentués en mode interactif suivant notre plateforme.
- Si les commandes sont lues depuis un fichier, la situation est légèrement différente : en général, nous pourrons utiliser des caractères accentués, mais ils risquent d'être interprétés différemment si nous transférons nos fichiers entre des plateformes différentes.

Le type str

Longueur

- La commande `len` permet de calculer la longueur d'une chaîne de caractères.
- Exemple :

```
>>> a='anticonstitutionnellement'
>>> len(a)
25
```

Appartenance à une chaîne

- La fonction `in` permet de tester l'appartenance d'un élément à une chaîne :
- Exemples :
  - `>>> 'z' in a`
    `False`
  - `>>> 'nt' in a`
    `True`

Concaténation et répétition

- Contrairement aux listes, les chaînes de caractères ne sont pas modifiables.
- Les chaînes peuvent être concaténées (accolées) avec l'opérateur $+$ et répétées avec $*$.
- Exemples :

```
>>> mot='Help'+' '+'me'
>>> print(mot)
Help me
>>> print(mot*3)
Help meHelp meHelp me
>>> print((mot+' ')*3)
Help me Help me Help me
```

Opérations sur les chaînes

Slicing

- Les chaînes peuvent être décomposées (indexées), comme en C. Le premier caractère d'une chaîne est en position (index) 0. Il n'y a pas de type spécifique aux caractères; un caractère est simplement une chaîne de taille un. Les sous-chaînes peuvent être spécifiées par slicing.
- Exemples :
  - `>>> mot='help me'`
    `>>> print(mot[4])`
  - `>>> print(mot[5])`
    `m`
  - `>>> mot[0:2]`
    `'he'`
  - `>>> mot[2:5]`
    `'lp '`
  - `>>> mot[:2]`
    `'he'`
  - `>>> mot[2:]`
    `'lp me'`

Slicing (suite et fin)

- Les indices de découpage erronés sont gérés de façon élégante: un index qui est trop grand est remplacé par la taille de la chaîne, un index de fin inférieur à l'indice de début retourne une chaîne vide.
- Exemple :

```
>>> mot[1:100]
'elp me'
```

```
>>> mot[10:]
''
>>> mot[2:1]
''
```

- Exemples :

```
>>> y='Please '+mot[1:]
>>> print(y)
Please help me
```

Itération sur une chaîne

Les chaînes sont des itérables

- En Python, les chaînes de caractères sont des itérables, c'est-à-dire que l'on peut les utiliser dans une boucle `for`.
- Exemple :

```
>>> nom='ABCDE'
>>> for a in nom:
print('a')
```

```
A
B
C
D
E
```

Recherche d'un mot

Le problème

- Soient u et t deux chaînes de caractères. Il s'agit de trouver un algorithme renvoyant *non* si u n'est pas une sous-chaîne de t, et i, position de la première occurence de u dans t sinon.
- Ce problème se rencontre dans de nombreux domaine (traitements de texte, génétique, etc).
- Nous présentons ici un algorithme naïf. Il existe de nombreuses autres méthodes : algorithmes de Rabin-Karp, KPM (Knuth-Morris-Pratt), de Boyer-Moore, etc.

Algorithme naïf

Données : Deux chaînes de caractères t et u;
Résultat : non si u n'est pas une sous-chaîne de t, la position de la première occurrence de u dans t sinon;
Initialisation : n $\leftarrow$ longueur de t, m $\leftarrow$ longueur de u;
pour $0 \leqslant i \leqslant n\text{-}m$ **faire**
    j $\leftarrow$ 0;
    tant que $j < m$ *et* $t[i+j]=u[j]$ **faire**
        j $\leftarrow$ j+1;
    si $j = m$ **alors**
        Renvoyer i;
Renvoyer non;

Recherche d'un mot

Terminaison et correction

- La boucle *while* s'arrête car dans le pire des cas j atteint la valeur m.
- Il est clair que, pour tout i, `u[0:j+1]=t[i:i+j+1]` est un invariant de la boucle *while* (la propriété est vérifiée au début de chaque itération). On en déduit la correction de l'algorithme.

Complexité

- Dans le meilleur des cas, on trouve clairement m.
- Dans le pire des cas, on trouve $(n - m + 1)m$.