Chaînes de caractères

Skander Zannad et Judicaël Courant

Lycée La Martinière-Monplaisir

2013-11-11

Plan

1 Manipulation des chaînes

1.1 Manipulations élémentaires

Caractère : lettre, symbole, blanc, retour à la ligne...

Chaîne de caractères : suite de caractères (grosso modo : tableau de caractères).

En python :

- pas d'objet spécifique de type «caractère».
- des objets de type chaîne (`str`).

Constantes chaînes :

```
'hello world!'
"hello world!"
'''hello world'''
"""hello world"""
```

Très proche des tableaux en termes de manipulation :

- Concaténation de deux chaînes `x` et `y` : `x + y`

  `x * n` : `x + ... + x` (`n` fois `x`).
- Accès à une sous-chaîne (slicing) : `x[i:j]` (chaîne des caractères d'indices appartenant à [[`i`,`j`[[).
- Mais :
  - `x[i]` : chaîne constituée d'un seul caractère, celui d'indice `i`.
  - Les chaînes ne sont pas modifiables (`a[i]='t'` interdit).

1.2 Méthodes utiles

En python tout est objet. Un objet est l'association de données [1] et de fonctions associées à l'objet appelées *méthodes* de l'objet.

Obtenir la liste des méthodes de `x` : `dir(x)`.

Pour les chaînes, on trouve notamment :

1. Appelées *attributs* de l'objet.

- `isalpha`, `isdigit`, `isspace`, `islower`, `isupper` : `True` ssi la chaîne est non-vide et que tous ses caractères sont des lettres (resp. des chiffres, des espaces, que toutes ses lettres sont des minuscules, que toutes ses lettres sont des majuscules). Exemple :

```
>>> 'Hello world!'.isalpha()
False
>>> 'the answer is 42'.islower()
True
```

- `lower`, `upper` : change en minuscule (resp majuscule) chaque lettre :

```
>>> 'Hello world!'.lower()
'hello world!'
```

- `join` :

```
>>> ';'.join(['alice', 'bob', 'charlie'])
'alice;bob;charlie'
```

- `split` :

```
>>> "What's new pussycat?".split()
["What's", 'new', 'pussycat?']
```

- `find` : recherche d'une sous-chaîne dans la chaîne :

```
>>> "anticonstitutionnellement".find('on')
5
>>> "anticonstitutionnellement".find('bon')
-1
```

– `count` : nombre d'occurrences d'une sous-chaîne dans une chaîne :

```
>>> "anticonstitutionnellement".count('on')
2
>>> "anticonstitutionnellement".count('bon')
0
```

1.3 Algorithme de recherche d'une sous-chaîne

- Fournie par la méthode `find`.
- Mais l'algorithme est explicitement au programme.

Idée simple : On décompose le problème en deux.

- une fonction `cherche(x, s)` cherchant `x` dans `s` et retournant le plus petit `i` tel que `x` soit une sous-chaîne de `s` commençant à l'indice `i`
- une fonction `est_sous_chaine(x, s, i)` testant si `x` est une sous-chaîne de `s` commençant à l'indice `i` (autrement dit : `x` est un *préfixe* de `s[i:]`).

Réalisation :

```
def cherche(s, x):
    """Plus petit i tel que x soit un préfixe de s[i:].
       -1 si x n'apparaît pas dans s."""
    for i in len(s):
        if est_sous_chaine(x, s, i):
            return i
    return -1
```

NB : dans certains langages de programmation, pas de return au milieu d'une boucle (seulement à la fin). Dans ce cas, on peut écrire la fonction `cherche` de la façon suivante :

```
def cherche(s, x):
    """Plus petit i tel que x soit un préfixe de s[i:].
       -1 si x n'apparaît pas dans s."""
    r = -1
    for i in len(s):
        if est_sous_chaine(x, s, i):
            r = i
            break
    return r
```

Dans certains langages, pas de **break**. On peut s'en passer :

```
def cherche(s, x):
    i = 0
    while i < len(s) and not(est_sous_chaine(x, s, i)):
        i += 1
    # ici ou bien i == len(s)
    # ou bien (i<len(s) et est_sous_chaine(x, s, i))
    if i < len(s):
        r = -1
    else:
        r = i
    return r
```

NB :

- invariant de la boucle : `i <= len(s)` et pour tout `k<i`, `s` ne contient pas la sous-chaîne `x` commençant à l'indice `k`.
- variant : `len(s)` $- 1$

2 Chaînes : le poids de l'histoire

2.1 ASCII

1960s : naissance du standard ASCII pour représenter les caractères.

Caractère : codé par un entier dans $[\![0, 128[\![$.

Peut être codé sur 7 bits, donc sans problème sur un octet.

Une chaîne est juste un tableau d'octets en mémoire. Adresse mémoire du caractère d'indice i : $s + i$ où s est l'adresse du début de la chaîne (donc accès en $O(1)$).

Très vite adopté universellement.

Notation hexadécimale des octets

Un octet $= 2 \times$ 4 bits.

4 bits : 16 possibilités, représentées par un chiffre *hexadécimal* (chiffre de 0 à 9 ou lettre de a à f)

`0 0000`	`4 0100`	`8 1000`	`c 1100`
`1 0001`	`5 0101`	`9 1001`	`d 1101`
`2 0010`	`6 0110`	`a 1010`	`e 1110`
`3 0011`	`7 0111`	`b 1011`	`f 1111`

Exemple : 5b représente l'octet 01011011.

```
|00 nul|01 soh|02 stx|03 etx|04 eot|05 enq|06 ack|07 bel|
|08 bs |09 ht |0a nl |0b vt |0c np |0d cr |0e so |0f si |
|10 dle|11 dc1|12 dc2|13 dc3|14 dc4|15 nak|16 syn|17 etb|
|18 can|19 em |1a sub|1b esc|1c fs |1d gs |1e rs |1f us |
|20 sp |21  ! |22  " |23  # |24  $ |25  % |26  & |27  ' |
|28  ( |29  ) |2a  * |2b  + |2c  , |2d  - |2e  . |2f  / |
|30  0 |31  1 |32  2 |33  3 |34  4 |35  5 |36  6 |37  7 |
|38  8 |39  9 |3a  : |3b  ; |3c  < |3d  = |3e  > |3f  ? |
|40  @ |41  A |42  B |43  C |44  D |45  E |46  F |47  G |
|48  H |49  I |4a  J |4b  K |4c  L |4d  M |4e  N |4f  O |
|50  P |51  Q |52  R |53  S |54  T |55  U |56  V |57  W |
|58  X |59  Y |5a  Z |5b  [ |5c  \ |5d  ] |5e  ^ |5f  _ |
|60  ` |61  a |62  b |63  c |64  d |65  e |66  f |67  g |
|68  h |69  i |6a  j |6b  k |6c  l |6d  m |6e  n |6f  o |
|70  p |71  q |72  r |73  s |74  t |75  u |76  v |77  w |
|78  x |79  y |7a  z |7b  { |7c  | |7d  } |7e  ~ |7f del|
```

A comme ...

ASCII : American Standard Code for Information Interchange.

«American» au sens de «États-Unien» :

- sans l'Amérique latine (español, português)
- ni le Québec (français)

(aucune lettre accentuée)

2.2 Les rustines

- Octet : 256 valeurs possibles donc 256 caractères codables.
- ASCII : 128 valeurs
- 128 places restantes $\rightarrow$ norme ISO 8859-1.
- Ne suffit pas pour l'Europe Centrale $\rightarrow$ ISO 8859-2 (différent).
- Et pour le turc et le maltais et l'esperanto ? ISO 8859-3
- Et les pays baltes ? ISO 8859-4
- Au fait, en russe, on utilise pas un alphabet bizarre ? ISO 8859-5.
- Et en arabe ? ISO 8859-6
- Et en grec ? ISO 8859-7
- Et l'hébreu ? ISO 8859-8

Actuellement, on en est à l'ISO 8859-16.

- Quelques quasi-doublons (ISO 8859-15 $\approx$ ISO 8859-1 + €)
- La plupart n'ont en commun que la partie ASCII.
- Quid des langues orientales ?

Microsoft : ses propres codages (proches des ISO mais différents) :

- CP437, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861, CP862, CP863, CP865, CP866, CP869, CP872.
- Windows-1250, Windows-1251, Windows-1252, Windows-1253, Windows-1254, Windows-1255, Windows-1256, Windows-1257, Windows-1258

Et si on reprenait le problème au début ?

2.3 Unicode

Fin 1980s/début 1990s : idée de standardiser un codage universel.

En 2012, $1,1 \times 10^5$ «caractères» définis, couvrant 100 écritures.

Chacun est repéré par un «point de code» (*code point*), entier de $[\![0, 1\,114\,112[\![$.

Ne tient pas sur un octet, ni même sur deux.

UTF-32

Codage de l'unicode sur 32 bits.

Inconvénient majeur : incompatibilité avec l'ASCII. Un texte ASCII n'est pas un texte UTF-32 valide. Il y a des quantités énormes de textes écrites en ASCII.

$\rightarrow$ UTF-32 n'est quasiment pas utilisé pour le stockage sous forme de fichiers.

UTF-8

- Codage de taille variable : 1 à 4 octets.
- Compatible ASCII : un caractère ASCII $\rightarrow$ un octet, codage inchangé.
- Actuellement le plus utilisé pour les pages web.
- Format canonique des fichiers textes sur toutes les plateformes (sauf Microsoft).
- Disponible sur toutes les plateformes, y compris Microsoft.

UTF-8 ou UTF-32 ?

Problèmes de l'UTF-8 :

- Une chaîne de 3 octets peut représenter un texte de 1, 2 ou 3 caractères : le calcul de la longueur nécessite de regarder le contenu, donc n'est plus en $O(1)$.
- Calcul du i^{e} élément : plus en temps constant ?

Alors qu'en UTF-32 :

- Longueur : nombre d'octets divisé par 4 (calcul en $O(1)$).
- i^{e} élément à l'adresse $s + 4i$ où s adresse du début.

UTF-8 et UTF-32

Conclusion :

- UTF-8 pour le stockage dans les fichiers.
- UTF-32 pour la représentation en mémoire.
- À chaque fois qu'on lit/écrit un fichier, on convertit.

3 Retour sur les chaînes Python

On restera sur Python 2.x. Ce qui suit est valable pour Python 2.x, pas pour Python 3.x.

3.1 Type `str`

Type des chaînes de caractères : `str`.

Historiquement :

$$\text{chaîne de caractères} \;=\; \text{suite d'octets}$$

Lecture fichier texte : lignes stockées dans une chaîne de caractères.

Lecture fichier binaire : données stockées dans une chaîne de caractères.

3.2 `str` : chaîne d'octets (pas de caractères)

On peut mettre n'importe quelle suite d'octets dans une chaîne. Par exemple, pour mettre les octets 68, $c3$ et $a9$ dans une chaîne :

```
s = '\x68\xc3\xa9'
```

À l'affichage, octets ASCII imprimés normalement. Octets non-ASCII interprétés selon le codage choisi par python (fonction de la configuration de l'OS/de python). Par exemple avec UTF-8 :

```
>>> print(s)
hé
```

3.3 Travailler avec des textes

Deux possibilités pour travailler avec les chaînes de caractères en Python 2 :

- Travailler avec les suites d'octets (`str`) et ignorer les problèmes de lettres accentuées. Problèmes (mineurs) à prévoir...
- Utiliser le type `unicode` (chaînes unicode).

Utilisation d'unicode

Conversion des chaînes d'octets en unicode :

```
t = s.decode("utf-8")
```

Conversion unicode vers chaîne d'octets :

```
x = t.encode("utf-8")
```

Constante chaîne unicode :

```
u'chaîne'
```

Précautions d'usage

- Les opérations vue section 1 sur `str` existent également en unicode.
- Ne pas mélanger `unicode` et `str`.

Règle à respecter : travailler le plus possible avec `unicode` et le moins possible avec `str`. En particulier :

- Convertir dès que possible les données lues en unicode.
- Convertir le plus tard possible les données à écrire en suite d'octets.

3.4 En Python 3

Les concepteurs de Python 3 ont fait les choses proprement :

- Chaînes (type `str`) : vraies chaînes unicode.
- Nouveau type `bytes` pour les suite d'octets.
- Lectures dans un fichier : retournent des `bytes` si ouvert en mode binaire, des `str` si mode texte (conversion automatique).
- Écriture : même principe.

Malheureusement, à de nombreux égards Python 3 est encore jeune, donc on restera pour l'instant sur Python 2.