Exemples_Dictionnaires_Bytes

December 15, 2020

0.1 Ce fichier est un notebook Python.

Il comporte deux types de cellules :

- les cellules d'édition dans lesquelles vous pouvez saisir du texte éventuellement enrichi de mises en formes ou de liens hypertextes avec la syntaxe du langage HTML simplifié qui s'appelle Markdown. Voir http://daringfireball.net/projects/markdown/ pour la syntaxe de Markdown.
- les cellules de code où l'on peut saisir du code Python3 puis le faire exécuter avec la combinaison de touches `CTRL + RETURN`

Une cellule peut être éditée de deux façons différentes :

- en mode *commande* lorsqu'on clique sur sa marge gauche qui est surlignée alors en bleu, on peut alors :
  - changer le type de la cellule en appuyant sur `m` pour passer en cellule Markdown ou sur `y` pour passer en cellule de code
  - insérer une cellule juste au-dessus en appuyant sur `a`
  - insérer une cellule juste en-dessous en appuyant sur `b`
  - couper la cellule en appuyant sur `x` etc …
- en mode *édition* lorsqu'on clique sur l'intérieur de la cellule.

L'aide complète sur les raccourcis claviers est accessible depuis le bouton `Help` dans la barre d'outils ci-dessus.

1 Définition

- Un **dictionnaire** est une structure de données qui est un tableau associatif associant des *valeurs* à des *clefs*. L'exemple type d'utilisation d'un dictionnaire est celui de l'annuaire téléphonique.
- En Python cette structure de données est caractérisée par le type `dict`.

  Premier exemple de définition de dictionnaire (ajouts successifs) :

```
annuaire = dict()   #ou dico = {}
annuaire['anna'] = '0478405030'
annuaire['eve'] = '0666666666'
annuaire['adam'] = '0666666666'
```

Second exemple de définition de dictionnaire (extension) :

```
annuaire = {'anna' : '0478405030',  'eve' : '0666666666',  'adam' :  '0666666666'}
```

Troisième exemple de définition de dictionnaire (compréhension) :

```
annuaire = {personne : telephone for (personne, telephone) in [('anna', '0478405030'),
```

Quatrième exemple de définition de dictionnaire (conversion d'une liste de tuples (clef, valeur)) :

```
annuaire = dict([('anna', '0478405030'),('eve', '0666666666'),  ('adam',   '0666666666')])
```

- Contrairement à un tableau indexé, les valeurs ne sont pas rangées de façon contigue en mémoire, il n'y a donc pas de notion d'ordre dans un dictionnaire.
- Par rapport à un tableau, un dictionnaire offre une meilleure complexité temporelle pour l'opération de test d'appartenance mais au prix d'un coût d'occupation mémoire bien plus important.

Opération	Dictionnaire	Tableau indexé
Accès / modification	constant O(1)	constant O(1)
Test d'appartenance	constant O(1)	linéaire O(n)

2 Exemples de définition d'un dictionnaire

Premier exemple :

[14]:
```
annuaire = dict()   #ou dico = {}
annuaire['anna'] = '0478405030'
annuaire['eve'] = '0666666666'
annuaire['adam'] = '0666666666'
print("annuaire de type : ", type(annuaire))
print(annuaire)
```

```
annuaire de type :  <class 'dict'>
{'anna': '0478405030', 'eve': '0666666666', 'adam': '0666666666'}
```

Second exemple :

[15]:
```
annuaire = {'anna' : '0478405030',  'eve' : '0666666666',  'adam' : ␣
 ↪'0666666666'}
print("annuaire de type : ", type(annuaire))
print(annuaire)
```

```
annuaire de type :  <class 'dict'>
{'anna': '0478405030', 'eve': '0666666666', 'adam': '0666666666'}
```

Troisième exemple :

[16]:
```
annuaire = {personne : telephone for (personne, telephone) in [('anna',␣
↪'0478405030'),
                                                               ('eve',␣
↪'0666666666'),  ('adam',  '0666666666')]}
print("annuaire de type : ", type(annuaire))
print(annuaire)
```

```
annuaire de type :  <class 'dict'>
{'anna': '0478405030', 'eve': '0666666666', 'adam': '0666666666'}
```

3 Opérations de bases sur un dictionnaire

3.1 Ajout / Accès / Modification

[22]:
```
print(annuaire['anna'])   #accès à la valeur associée à la clef 'anna'
```

```
0478405030
```

[21]:
```
print(annuaire['donald'])  #erreur d'accès, clef manquante
```

```
      ␣
↪---------------------------------------------------------------------------

       KeyError                                  Traceback (most recent call␣
↪last)

       <ipython-input-21-0aa81ae9a096> in <module>
    ----> 1 print(annuaire['donald'])


       KeyError: 'donald'
```

[23]:
```
annuaire['assia'] = '0752545150'  #ajout d'une association (clef, valeur)
```

[24]:
```
print(annuaire)
```

```
{'anna': '0478405030', 'eve': '0666666666', 'adam': '0666666666', 'assia':
'0752545150'}
```

[25]:
```
print(len(annuaire))    #longueur du dictionnaire, comme pour le type `list`
```

```
4
```

[26]:
```
annuaire['assia'] = '0632343130'   #modification de la valeur associée à la␣
↪clef 'assia'
```

[27]:
```
print(annuaire)
```

```
{'anna': '0478405030', 'eve': '0666666666', 'adam': '0666666666', 'assia':
'0632343130'}
```

4 Parcours de dictionnaire

Premier parcours : sur les clefs :

[29]:
```
for clef in annuaire:
    print(clef)
```

```
anna
eve
adam
assia
```

Une alternative :

[31]:
```
for clef in annuaire.keys():
    print(clef)
```

```
anna
eve
adam
assia
```

Second parcours : sur les valeurs :

[33]:
```
for valeur in annuaire.values():
    print(valeur)
```

```
0478405030
0666666666
0666666666
0632343130
```

Troisième parcours : sur les couples (clef, valeur)

[34]:
```
for clef, valeur in annuaire.items():
    print(clef, valeur)
```

```
anna 0478405030
eve 0666666666
adam 0666666666
assia 0632343130
```

4.1 Et dans le cadre du mini-projet ?

Prenons l'exemple d'un dictionnaire qui associe à tous les caractères ASCII (d'ordinal < 128), leur code ASCII en hexadécimal

[39]:
```
table = {chr(k) : hex(k) for k in range(128)}
print(table)
```

```
{'\x00': '0x0', '\x01': '0x1', '\x02': '0x2', '\x03': '0x3', '\x04': '0x4',
'\x05': '0x5', '\x06': '0x6', '\x07': '0x7', '\x08': '0x8', '\t': '0x9', '\n':
'0xa', '\x0b': '0xb', '\x0c': '0xc', '\r': '0xd', '\x0e': '0xe', '\x0f': '0xf',
'\x10': '0x10', '\x11': '0x11', '\x12': '0x12', '\x13': '0x13', '\x14': '0x14',
'\x15': '0x15', '\x16': '0x16', '\x17': '0x17', '\x18': '0x18', '\x19': '0x19',
'\x1a': '0x1a', '\x1b': '0x1b', '\x1c': '0x1c', '\x1d': '0x1d', '\x1e': '0x1e',
'\x1f': '0x1f', ' ': '0x20', '!': '0x21', '"': '0x22', '#': '0x23', '$': '0x24',
'%': '0x25', '&': '0x26', "'": '0x27', '(': '0x28', ')': '0x29', '*': '0x2a',
'+': '0x2b', ',': '0x2c', '-': '0x2d', '.': '0x2e', '/': '0x2f', '0': '0x30',
'1': '0x31', '2': '0x32', '3': '0x33', '4': '0x34', '5': '0x35', '6': '0x36',
'7': '0x37', '8': '0x38', '9': '0x39', ':': '0x3a', ';': '0x3b', '<': '0x3c',
'=': '0x3d', '>': '0x3e', '?': '0x3f', '@': '0x40', 'A': '0x41', 'B': '0x42',
'C': '0x43', 'D': '0x44', 'E': '0x45', 'F': '0x46', 'G': '0x47', 'H': '0x48',
'I': '0x49', 'J': '0x4a', 'K': '0x4b', 'L': '0x4c', 'M': '0x4d', 'N': '0x4e',
'O': '0x4f', 'P': '0x50', 'Q': '0x51', 'R': '0x52', 'S': '0x53', 'T': '0x54',
'U': '0x55', 'V': '0x56', 'W': '0x57', 'X': '0x58', 'Y': '0x59', 'Z': '0x5a',
'[': '0x5b', '\\': '0x5c', ']': '0x5d', '^': '0x5e', '_': '0x5f', '`': '0x60',
'a': '0x61', 'b': '0x62', 'c': '0x63', 'd': '0x64', 'e': '0x65', 'f': '0x66',
'g': '0x67', 'h': '0x68', 'i': '0x69', 'j': '0x6a', 'k': '0x6b', 'l': '0x6c',
'm': '0x6d', 'n': '0x6e', 'o': '0x6f', 'p': '0x70', 'q': '0x71', 'r': '0x72',
's': '0x73', 't': '0x74', 'u': '0x75', 'v': '0x76', 'w': '0x77', 'x': '0x78',
'y': '0x79', 'z': '0x7a', '{': '0x7b', '|': '0x7c', '}': '0x7d', '~': '0x7e',
'\x7f': '0x7f'}
```

On peut aussi donner les codes ASCII sous forme d'octets (type `bytes`), mais il faut alors convertir les représentations entiers `k` en hexadécimal, puis leur appliquer la fonction `bytes.fromhex` mais celle-ci attend un nombre de deux chiffres en hexadécimal. Il faut donc enlever le préfixe `0x` avec la méthode `lstrip` puis remplir éventuellement par des '0' à gauche avec `zfill`.

[51]:
```
table2 = {chr(k) : bytes.fromhex(hex(k).lstrip('0x').zfill(2)) for k in␣
 ↪range(128)}
print(table2)
```

```
{'\x00': b'\x00', '\x01': b'\x01', '\x02': b'\x02', '\x03': b'\x03', '\x04':
b'\x04', '\x05': b'\x05', '\x06': b'\x06', '\x07': b'\x07', '\x08': b'\x08',
'\t': b'\t', '\n': b'\n', '\x0b': b'\x0b', '\x0c': b'\x0c', '\r': b'\r', '\x0e':
b'\x0e', '\x0f': b'\x0f', '\x10': b'\x10', '\x11': b'\x11', '\x12': b'\x12',
'\x13': b'\x13', '\x14': b'\x14', '\x15': b'\x15', '\x16': b'\x16', '\x17':
b'\x17', '\x18': b'\x18', '\x19': b'\x19', '\x1a': b'\x1a', '\x1b': b'\x1b',
'\x1c': b'\x1c', '\x1d': b'\x1d', '\x1e': b'\x1e', '\x1f': b'\x1f', ' ': b' ',
'!': b'!', '"': b'"', '#': b'#', '$': b'$', '%': b'%', '&': b'&', "'": b"'",
```

```
'(': b'(', ')': b')', '*': b'*', '+': b'+', ',': b',', '-': b'-', '.': b'.',
'/': b'/', '0': b'0', '1': b'1', '2': b'2', '3': b'3', '4': b'4', '5': b'5',
'6': b'6', '7': b'7', '8': b'8', '9': b'9', ':': b':', ';': b';', '<': b'<',
'=': b'=', '>': b'>', '?': b'?', '@': b'@', 'A': b'A', 'B': b'B', 'C': b'C',
'D': b'D', 'E': b'E', 'F': b'F', 'G': b'G', 'H': b'H', 'I': b'I', 'J': b'J',
'K': b'K', 'L': b'L', 'M': b'M', 'N': b'N', 'O': b'O', 'P': b'P', 'Q': b'Q',
'R': b'R', 'S': b'S', 'T': b'T', 'U': b'U', 'V': b'V', 'W': b'W', 'X': b'X',
'Y': b'Y', 'Z': b'Z', '[': b'[', '\\': b'\\', ']': b']', '^': b'^', '_': b'_',
'`': b'`', 'a': b'a', 'b': b'b', 'c': b'c', 'd': b'd', 'e': b'e', 'f': b'f',
'g': b'g', 'h': b'h', 'i': b'i', 'j': b'j', 'k': b'k', 'l': b'l', 'm': b'm',
'n': b'n', 'o': b'o', 'p': b'p', 'q': b'q', 'r': b'r', 's': b's', 't': b't',
'u': b'u', 'v': b'v', 'w': b'w', 'x': b'x', 'y': b'y', 'z': b'z', '{': b'{',
'|': b'|', '}': b'}', '~': b'~', '\x7f': b'\x7f'}
```

Pour inverser la table précédente ((clef, valeur) -> (valeur, clef)) c'est très simple en combinant compréhension de dictionnaire et parcoursp par (clef, valeur) !

[53]:
```
table2_inv = {valeur : clef for (clef, valeur) in table2.items()}
print(table2_inv)
```

```
{b'\x00': '\x00', b'\x01': '\x01', b'\x02': '\x02', b'\x03': '\x03', b'\x04':
'\x04', b'\x05': '\x05', b'\x06': '\x06', b'\x07': '\x07', b'\x08': '\x08',
b'\t': '\t', b'\n': '\n', b'\x0b': '\x0b', b'\x0c': '\x0c', b'\r': '\r',
b'\x0e': '\x0e', b'\x0f': '\x0f', b'\x10': '\x10', b'\x11': '\x11', b'\x12':
'\x12', b'\x13': '\x13', b'\x14': '\x14', b'\x15': '\x15', b'\x16': '\x16',
b'\x17': '\x17', b'\x18': '\x18', b'\x19': '\x19', b'\x1a': '\x1a', b'\x1b':
'\x1b', b'\x1c': '\x1c', b'\x1d': '\x1d', b'\x1e': '\x1e', b'\x1f': '\x1f', b'
': ' ', b'!': '!', b'"': '"', b'#': '#', b'$': '$', b'%': '%', b'&': '&', b"'":
"'", b'(': '(', b')': ')', b'*': '*', b'+': '+', b',': ',', b'-': '-', b'.':
'.', b'/': '/', b'0': '0', b'1': '1', b'2': '2', b'3': '3', b'4': '4', b'5':
'5', b'6': '6', b'7': '7', b'8': '8', b'9': '9', b':': ':', b';': ';', b'<':
'<', b'=': '=', b'>': '>', b'?': '?', b'@': '@', b'A': 'A', b'B': 'B', b'C':
'C', b'D': 'D', b'E': 'E', b'F': 'F', b'G': 'G', b'H': 'H', b'I': 'I', b'J':
'J', b'K': 'K', b'L': 'L', b'M': 'M', b'N': 'N', b'O': 'O', b'P': 'P', b'Q':
'Q', b'R': 'R', b'S': 'S', b'T': 'T', b'U': 'U', b'V': 'V', b'W': 'W', b'X':
'X', b'Y': 'Y', b'Z': 'Z', b'[': '[', b'\\': '\\', b']': ']', b'^': '^', b'_':
'_', b'`': '`', b'a': 'a', b'b': 'b', b'c': 'c', b'd': 'd', b'e': 'e', b'f':
'f', b'g': 'g', b'h': 'h', b'i': 'i', b'j': 'j', b'k': 'k', b'l': 'l', b'm':
'm', b'n': 'n', b'o': 'o', b'p': 'p', b'q': 'q', b'r': 'r', b's': 's', b't':
't', b'u': 'u', b'v': 'v', b'w': 'w', b'x': 'x', b'y': 'y', b'z': 'z', b'{':
'{', b'|': '|', b'}': '}', b'~': '~', b'\x7f': '\x7f'}
```