Chapitre 1.4 : Chaînes de caractères et tuples

I Chaînes de caractères

On a déjà utilisé des **chaînes de caractères** (string en anglais, `str`) dans la fonction `print`. C'est le type qui permet de représenter des textes :

- on crée un objet de type `str` en écrivant un texte entre guillemets (ou entre apostrophes). La taille de la chaîne s'obtient grâce à la fonction `len` : c'est le nombre de caractères qui constituent la chaîne (espaces compris). Par exemple :

```
>>> s = 'Bon jour'
>>> len(s)
8
```

- les caractères d'une chaîne sont numérotés de `0` à `len(s) - 1`. On peut accéder à un caractère de la chaîne à l'aide de `s[i]`, où `i` est la position du caractère. Attention à ne pas dépasser !

```
>>> s[3]
' '
>>> s[8]
IndexError: string index out of range
```

- On peut aussi accéder à un élément en comptant à partir de la fin : `s[-1]` est le dernier élément, `s[-2]` l'avant dernier et ainsi de suite jusqu'à `s[-len(s)]` qui est le premier.
- On peut aussi récupérer une sous-chaîne de caractères : `s[i:j]` renvoie la chaîne de caractères composée des caractères `s[i]`, `s[i+1]`, ..., `s[j-1]`. On peut aussi utiliser un pas : `s[i:j:pas]` de sorte que `s[0:len(s):2]` (ou de façon équivalente `s[::2]`) renvoie la chaîne constituée d'un caractère sur 2 de `s`. On dit que l'on fait des **tranches** (slicing en anglais).

```
>>> s[:3]
'Bon'
>>> s[::-1] #on parcours à l'envers
'ruoj noB'
```

- Les chaînes de caractères sont **immuables** : on ne peut pas modifier de caractère.

```
>>> s[0] = 'b'
TypeError: 'str' object does not support item assignment
```

- On peut concaténer deux chaînes de caractères grâce à `+` et répéter une chaîne en utilisant `*`. Attention, ces deux opérateurs créent une nouvelle chaîne de caractères.

```
>>> t = " !"
>>> s[:3] + s[4:] +t
'Bonjour !'
>>> s+3*t
'Bon jour ! ! !'
```

- On peut itérer sur les caractères d'une chaîne dans une boucle `for`.

```
for c in s:
    instructions...
```

- On peut convertir des nombres en chaînes de caractères et vice-versa.

```
>>> pi, Pi = 3.14, str(3.14)
>>> 2*pi
6.28
>>> 2*Pi
'3.143.14'
>>> E = '2.7'
>>> e = float(E)
```

```
>>> e+pi
5.84
```

II tuples

Un **tuple** est une suite d'éléments quelconques séparés par une virgule et éventuellement encadrés par des parenthèses. Le tuple vide est défini par `()` et un singleton par `(a,)`. Par exemple

```
>>> u = (1, 'B', True, 0.2)
```

La manipulation des tuples est la même que pour les chaînes de caractères. En particulier, un tuple est **immuable**, et on peut itérer dessus.

```
>>> len(u)
4
>>> u[1: 3]
('B', True)
```

On peut aussi déconstruire un tuple en affectant simultanément ses composantes à différentes variables.

```
>>> (x,y,z,t) = u
>>> z
True
```

Les chaînes de caractères et les tuples sont deux exemples de **séquences** : ce sont des types d'objets qui possèdent une **taille**, qui sont **itérables** et **indexables**. On verra un troisième type de séquence au prochain cours : les listes.

III Exercices

Exercice 1 On n'oubliera pas de commenter son script !

1. Écrire une fonction `prefixes(s)` qui affiche tous les préfixes d'un mot en utilisant du slicing. Par exemple,

```
>>> prefixes("PCSI2")

P
PC
PCSI
PCSI2
```

2. Écrire une fonction `palindrome(s)` qui teste si la chaîne de caractères `s` est un palindrome. La fonction doit renvoyer un booléen.
   On pourra comparer la chaîne `s` à la chaîne retournée grâce à un slicing.
3. On considère la fonction suivante :

```
def mystere(s):
    c = 0
    for e in s:
        if e == 'a' or e == 'e' or e = 'i' or e == 'o' or e == 'u' or e == 'y':
            c = c + 1
    return c
```

   (a) Quel est le type de `s` ?
   (b) Comment peut-on qualifier la variable `c` ?
   (c) Que fait la fonction `mystere` ?

■

Exercice 2 Écrire une fonction `element(e, t)` qui prend un objet `e` et un tuple `t` et qui renvoie `True` si `e` est dans `t` et `False` sinon. ■