1 Chaînes de caractères

Le type **str** (string) est celui des chaînes de caractères. Ce type permet de représenter les textes.

• Pour représenter une chaîne de caractères, on utilise des apostrophes ou des guillemets. Les guillemets sont utiles pour représenter des chaînes de caractères contenant des apostrophes.

```
>>> ch1 = 'bonjour'
>>> ch2 = "aujourd'hui"
```

• La commande **print** affiche la chaîne de caractères sans les apostrophes.

```
>>> ch = 'Hello'
>>> print(ch)
Hello
>>> print('Le résultat est', 3, 'après 10 itérations')
Le résultat est 3 après 10 itérations
```

• On obtient la longueur d'une chaîne de caractères avec la fonction **len**.

```
>>> len('Bonjour')
7
```

• Si n est la longueur de la chaîne de caractères, chaque caractère a un indice entre 0 et $n-1$, on accède au caractère d'indice i d'une chaîne ch avec la notation $ch[i]$. On peut aussi récupérer une tranche d'une chaîne de caractères comme avec les listes.

```
>>> ch = 'Bonjour à tous'
>>> ch[0]
'B'   # le résultat est une chaîne de caractères
>>> ch[8]
' '
>>> ch[-1]
's'
>>> ch[1 : 4]
'onj'     # le caractère d'indice 4 n'est pas pris
```

• Les chaînes de caractères peuvent être concaténées avec l'opérateur $+$ et répétées avec l'opérateur $*$:

```
>>> ch = 'abc' + 'def'
>>> print(ch)
abcdef
>>> ch = 'Non ! ' * 3
>>> print(ch)
Non ! Non ! Non !
```

• Contrairement aux listes, les chaînes de caractères ne sont pas modifiables.

```
>>> ch = 'bonjour'
>>> ch[0] = 'B'
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Par contre, le script ci-dessous fonctionne :

```
>>> ch = 'bonjour'
>>> ch = 'B' + ch[1:]
>>> print(ch)
'Bonjour'
```

• Comme pour une liste, il est tout à fait possible de parcourir une chaîne de caractères avec l'instruction *for i in ch*. Voici, à titre d'exemple, une fonction qui cherche si la lettre *b* est présente dans une chaîne de caractères.

```
def cher_b(ch):
    """renvoie True si la lettre b est présente dans la chaine de caractères et False sinon"""
    for i in ch:
        if i == 'b':
            return(True)
    return(False)
```

Bien entendu, le test $'b'$ *in ch* nous donne directement la réponse attendue.

2 Tuples

Un tuple est une collection ordonnée de plusieurs éléments. Les tuples ressemblent aux listes mais on ne peut pas les modifier une fois créés.

• Cela correspond aux n-uplets que l'on étudie en mathématiques, on utilise des parenthèses.

```
>>> a = (3, 1, 4)
>>> type(a)
<class 'tuple'>
```

• On peut aussi utiliser la notation suivante :

```
>>> x, y = 9, 13 # revient au même que (x, y) = (9, 13)
```

• Il est très simple de créer une fonction qui renvoie un tuple.

```
def test(x):
    return(x + 1, x - 1)
```

On utilise ensuite :

```
>>> a = test(7)
>>> a
(8, 6)
>>> b, c = test(7)
>>> b
8
>>> c
6
```

• La valeur d'un élément d'un tuple est obtenue en utilisant la même syntaxe que pour une liste. On peut aussi récupérer des tranches du tuple.

```
>>> a = (8, 3, 4)
>>> a[0]
8
>>> a[0:2]
(8, 3)
```

• Il est tout à fait possible de parcourir un tuple avec une boucle for. Par exemple :

```
a = (8, 3, 4)
for i in a:
    print(a)
```

Ce qui donne comme résultat :

```
8
3
4
```

• La concaténation fonctionne aussi :

```
>>> (1, 2) + (3, 4)
(1, 2, 3, 4)
```

• Quelles sont les différences entre une liste et un tuple ?
  - ▶ Les listes sont modifiables et les tuples ne sont pas modifiables.
  - ▶ Un tuple occupe une place mémoire plus petite que celle d'une liste, ce qui rend les manipulations plus rapides.
  - ▶ Les listes ont plus de fonctionnalités que les tuples.
  - ▶ Les tuples ont plutôt vocation à contenir des éléments de même type, on peut penser à des coordonnées dans le plan ou dans l'espace.

3 Dictionnaires

Beaucoup d'informations de la vie courante se présentent sous la forme d'associations de données. Par exemple, on souhaite représenter les menus des soirs d'une semaine : lundi : salade de pâtes, mardi : pizza...

Comment représenter ces correspondances de données ?

• Sous forme d'une liste de listes :

```
>>> L = [['Lundi', 'salade de pâtes'], ['Mardi', 'Pizza' ],...]
```

C'est assez lourd d'accéder aux données. En effet, pour savoir ce que l'on mange le jeudi soir, il faut trouver le numéro du jeudi dans la semaine et écrire $L[3][1]$.

• Sous la forme d'une simple liste :

```
>>> L = ['salade de pâtes', 'pizza ,...]
```

On perd l'association entre les jours de la semaine et les repas.

• La structure de donnée adaptée en Python pour répondre à ce problème est celle des dictionnaires, les indices ne seront pas forcément des numéros mais vont être, dans cet exemple, les jours de la semaine.

> Un dictionnaire va permettre de rassembler des éléments mais ceux-ci seront identifiés par une clé, qui peut être d'un type quelconque, au lieu d'être identifiés par un indice comme dans une liste.

• Pour créer un dictionnaire, on utilise des accolades en renseignant les couples *clé* : *valeur*. Par exemple, si l'on souhaite répertorier les menus des soirs d'une semaine :

```
>>> menus = {'lundi': 'salade de pâtes', 'mardi': 'pizza', 'mercredi': 'carottes', 'jeudi': 'raviolis', '
    vendredi': 'aubergines grillées', 'samedi': 'lasagnes', 'dimanche': 'soupe'}
```

Ici les clés sont les jours de la semaine, on peut accéder aux valeurs ainsi :

```
>>> menus['mardi']
'pizza'
```

• Il est très rapide d'ajouter un élément à un dictionnaire :

```
>>> menus['lundi2'] = 'tarte aux pommes de terre'
>>> menus
{'lundi': 'salade de pâtes', 'mardi': 'pizza', 'mercredi': 'carottes', 'jeudi': 'raviolis', 'vendredi': '
    aubergines grillées', 'samedi': 'lasagnes', 'dimanche': 'soupe', 'lundi2': 'tarte aux pommes de
    terre'}
```

Cet élément est ajouté à la fin mais il faut garder en tête que les entrées d'un dictionnaire ne sont pas ordonnées.

• Les clés d'un dictionnaire ne sont pas modifiables et sont uniques, c'est-à-dire que l'on ne peut pas avoir le dictionnaire $\{'lundi' : 'salade', 'lundi' : 'pizza'\}$. Par contre, les valeurs sont modifiables. Par exemple, en reprenant le dictionnaire *menus* :

```
>>> menus['jeudi'] = 'tourte au fromage'
>>> menus
{'lundi': 'salade de pâtes', 'mardi': 'pizza', 'mercredi': 'carottes', 'jeudi': 'tourte au fromage', '
    vendredi': 'aubergines grillées ', 'samedi': 'lasagnes', 'dimanche': 'soupe'}
```

• La construction par compréhension existe pour les dictionnaires :

```
>>> d = {x : x ** 2 for x in range(5)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

• On peut avoir accès au nombre d'éléments (les items) d'un dictionnaire avec la commande *len*.

• La méthode **items** permet d'obtenir l'ensemble des couples (*clé*, *valeur*). Ainsi, on peut parcourir un dictionnaire de la façon suivante :

```
>>> menus = {'lundi': 'salade de pâtes', 'mardi': 'pizza', 'mercredi': 'carottes', 'jeudi': ' raviolis ', '
    vendredi': 'aubergines grillées ', 'samedi': 'lasagnes', 'dimanche': 'soupe'}
>>> for elt in menus.items():
        print(elt)
('lundi', 'salade de pâtes')
('mardi', 'pizza')
('mercredi', 'carottes')
('jeudi', ' raviolis ')
('vendredi', 'aubergines')
('samedi', 'lasagnes')
('dimanche', 'soupe')
```

Les éléments renvoyés sont des tuples. On peut lister les valeurs :

```
>>> menus = {'lundi': 'salade de pâtes', 'mardi': 'pizza', 'mercredi': 'carottes', 'jeudi': ' raviolis ', '
    vendredi': 'aubergines grillées ', 'samedi': 'lasagnes', 'dimanche': 'soupe'}
>>> for elt in menus.items():
        print(elt [1])
salade de pâtes
pizza
carottes
 raviolis
aubergines
lasagnes
soupe
tarte
```

• Le mot clé **in** permet de tester l'appartenance d'une clé à un dictionnaire et non d'une valeur.

```
>>> 'carottes' in menus
False
>>> 'mercredi' in menus
True
```

• Comme pour les listes pour faire une copie indépendante, il est préférable d'utiliser la commande **deepcopy** du module *copy*.

```
>>> menus = {'lundi': 'salade de pâtes', 'mardi': 'pizza', 'mercredi': 'carottes', 'jeudi': 'raviolis', 'vendredi': 'aubergines grillées', 'samedi': 'lasagnes', 'dimanche': 'soupe'}
>>> import copy as c
>>> d = c.deepcopy(menus)
```

• Les commandes suivantes sont données à titre indicatif mais ne sont pas au programme.

```
>>> list(menus.keys()) # on récupère la liste des clés
['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi', 'dimanche']
>>> list(menus.values()) # on récupère la liste des valeurs
['salade de pâtes', 'pizza', 'carottes', 'raviolis', 'aubergines', 'lasagnes', 'soupe']
>>> del menus['lundi'] # supprime l'entrée qui correspond à la clé 'lundi'
>>> menus
{'mardi': 'pizza', 'mercredi': 'carottes', 'jeudi': 'raviolis', 'vendredi': 'aubergines', 'samedi': 'lasagnes', 'dimanche': 'soupe'}
>>> dict([['mardi', 'pizza'], ['mercredi', 'carottes']]) # crée un dictionnaire à partir d'une liste de listes ou d'une liste de couples
{'mardi': 'pizza', 'mercredi': 'carottes'}
>>> d.clear() # efface tous mes items du dictionnaire qui devient alors vide
>>> dico1.update(dico2) # étend dico1 en rajoutant les items de dico2
```

• Nous étudierons au cours de l'année des problèmes où les dictionnaires seront naturellement un bon choix de type pour les données. Mais de façon générale, l'usage d'un dictionnaire pourra être adapté lorsque l'on a une seule "entité" présentant plusieurs critères :

```
>>> profil = {'Nom' : 'Delmas', 'Prénom': 'Fabien', 'Age': 25, 'Ville': 'Rennes', 'Sexe': 'M'}
```

ou plusieurs entités ayant un critère en commun :

```
>>> LV2 = {'Anna': Allemand, 'Louis': Espagnol, 'Marthe': Espagnol, 'Léa': 'Russe', 'Chloé': 'Espagnol'}
```

• Nous verrons également que la recherche d'un élément dans un dictionnaire est plus rapide que la recherche d'un élément dans une liste.