NSI 1ère - Algorithmique - 5 - Tuples et dictionnaires

QK

Tuples et dictionnaires

p-uplets

- En mathématiques on connait les “couples” de valeurs :

  Le point A d'abscisse 3, d'ordonnée 4 se note $A(3;4)$.

- Quand on dispose de p données on parle de p-**uplet**.

  En anglais ça donne un “tuple”.

Les tuples en Python

- Dans Python les tuples sont des séries de données, séparées de virgules.

```
>>> un_tuple = (3, 4, 5)
>>> un_tuple[1] # comme pour une liste
4
>>> type(un_tuple)
<type 'tuple'>
>>> for x in un_tuple: # on peut itérer sur un tuple
      print(x**2)
9
16
25
>>> autre_tuple = ("pomme", 1, True)
```

Les tuples ne sont pas mutables !

```
>>> un_tuple[1] = 2 # changer un élément : NON

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assig
```

Une fois un tuple défini il n'est plus possible de le transformer (**non mutable**).

utiliser les tuples ?

- C'est comme une liste, mais plus rapide et pas modifiable.
- Pratique quand on sait qu'une liste de valeur ne changera plus.

Tuples et fonctions

- Une fonction peut renvoyer un tuple

```
>>> def f(x, y):
      return x**2, y**3
>>> f(3, 2)
(9, 8)
```

p-uplet nommés

- *Remarque importante* : ici le programme officiel de la spécialité NSI est imprécis :

  "En Python, les p-uplets nommés sont implémentés par des dictionnaires."

- C'est faux. Les dictionnaires en Python sont des objets particuliers et les "p-uplets nommés" sont implémentés par des *namedtuples* auxquels on accède via la librairie "collections"

- Je présenterai donc *les dictionnaires en Python* et pas les *namedtuples en Python*

  Nous parlerons toujours de dictionnaires et jamais plus de p-uplets nommés. Les idées dont nous avons besoin sont les mêmes.

et donc : les dictionnaires en Python

Les dictionnaires sont des “tableaux associatifs” indexés par des *clés*

```
>>> scores = {"Jean": 145, "Paul": 200, "Emy": 345 }
>>> scores["Emy"] = 364
>>> scores
{"Jean": 145, "Paul": 200, "Emy": 364 }
>>> del scores["Paul"]
>>> scores["Téo"] = 308
>>> scores
{"Jean": 145, "Emy": 364, "Téo": 308}
```

Les dictionnaires sont **mutables**. On accède à une **valeur** à l’aide de sa **clé**

Element d'un dictionnaire, conversions, tris

On peut tester l'appartenance, trier ou convertir un dictionnaire :

```
>>> scores = {"Jean": 145, "Emy": 364, "Téo": 308}
>>> "Jean" in scores
True
>>> list(scores)
["Jean", "Emy", "Téo"] # extraire la liste des clés
>>> sorted(scores)
["Emy", "Jean", "Téo"] # liste triée des clés
```

Dictionnaires par compréhension

On peut créer de plusieurs manières un dictionnaire par compréhension :

Par exemple, pour stocker les valeurs d'une fonction :

```
>>> {x: x**2 for x in  (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

On via une série d'associations :

```
>>> Quentin = dict(age=71, taille=180, poids=150)
>>> Quentin
{'age': 71, 'taille': 180, 'poids': 150}
```

Et avec `zip` :

```
>>> dict(zip(["x", "y", "z"], [18, 25, 32]))
{"x": 18, "y": 25, "z": 32}
```

Itérer dans un dictionnaire

On peut itérer dans un dictionnaire et récupérer les clés et les valeurs :

```
>>> persos = {'Robert': 'le musclé',
'Nadine': 'la rusée'}
>>> for key, value in persos.items():
        print(key, value)
Robert le musclé
Nadine la rusée
```

Clés et valeurs

On peut aussi itérer sur la liste des clés ou des valeurs :

```
>>> ingredients = {'poules': 2, 'coqs':4,
'cochons': 8}
>>> for x in ingredients.keys():
        print(x)
'poules'
'coqs'
'cochons'
>>> for y in ingredients.values():
        print(y)
2
4
8
```