T2.3 Dictionnaires

Jusqu'à présent, nous avons rencontré deux types de **conteneurs** (ou ensemble d'éléments) : les listes et les tuples. Ces deux types ont en commun le repérage de leurs éléments par un *indice*, qui est un nombre entier. Ces deux types sont **ordonnés**, on les appelle des **séquences**. L'indice est la **clé** qui permet d'accéder à un élément de l'ensemble.

Prenons comme exemple un autre ensemble d'éléments: les numéros de téléphone sauvegardés dans l'application « Contacts » de votre smartphone. Ces numéros ne sont pas ordonnés, et ne sont pas repérables par un *indice*, mais (en général) par un nom.

Grosso modo, le contenu de votre application « Contacts » est un ensemble d'associations `nom: numéro`. En Informatique, on parle de *p-uplet nommé*: un ensemble de (p) valeurs appelées par un descripteur autre qu'un indice.

En Python, on représentera ces *p-uplets nommés* par des **dictionnaires**.

1. 2.3.1 Définition d'un dictionnaire, clés et valeurs

Le type `dict`

Script Python

```
>>> contacts = {"jean-mi": '0155551010', "bénédicte": '0516526600', "marc":
'0545974500'}
```

- Un dictionnaire (type `dict`) est une donnée composite qui n'est pas ordonnée (à la différence des listes !). Il fonctionne par un système de `clé: valeur`.
- Les clés, comme les valeurs, peuvent être de types différents.
- Un dictionnaire est délimité par des accolades, les associations `clé: valeur` séparées par des espaces.
- On accède à une valeur par sa clé :

Script Python

```
>>> contacts["jean-mi"]
'0155551010'
```

Méthodes `keys` et `values`

On peut lister les clés d'un dictionnaires:

Script Python

```
>>> contacts.keys()
dict_keys(['jean-mi', 'bénédicte', 'marc'])
```

et ses valeurs:

Script Python

```
>>> contacts.values()
dict_values(['0155551010', '0516526600', '0545974500'])
```

Ajout/modification/suppression d'éléments

Pour modifier une valeur associée à une clé, on réaffecte la nouvelle valeur :

Script Python

```
>>> contacts["jean-mi"] = '0605040302'
>>> contacts
{'jean-mi': '0605040302', 'bénédicte': '0516526600', 'marc': '0545974500'}
```

Si la clé n'existe pas, cela ajoute une paire `clé: valeur` au dictionnaire:

Script Python

```
>>> contacts["xavier"] = '0545387000'
>>> contacts
{'jean-mi': '0605040302', 'bénédicte': '0516526600', 'marc': '0545974500', 'xavier':
'0545387000'}
```

La suppression d'une clé (et donc de sa valeur) se fait par le mot-clé `del`:

Script Python

```
>>> del contacts["jean-mi"]
>>> contacts
{'bénédicte': '0516526600', 'marc': '0545974500', 'xavier': '0545387000'}
```

2. 2.3.2 Méthodes sur les dictionnaires

Parcours d'un dictionnaire

On utilise comme pour les listes une boucle `for` avec l'opérateur `in`. Cet opérateur `in` peut également tester l'appartenance d'une clé à un dictionnaire, hors d'un `for`.

Méthode basique

On parcourt par défaut sur les clés:

Script Python

```
>>> for nom in contacts:
...     print("le numéro de {} est {}".format(nom, contacts[nom]))
...
le numéro de bénédicte est 0516526600
le numéro de marc est 0545974500
le numéro de xavier est 0545387000
```

Il serait équivalent de faire `for nom in contacts.keys()`.

On peut également parcourir le dictionnaire sur les valeurs, mais sans possibilité alors de récupérer la clé à partir de la valeur (puisque plusieurs clés peuvent avoir la même valeur).

On utilisera pour cela:

Script Python

```
>>> for v in contacts.values():
...     print(v)
...
0516526600
0545974500
0545387000
```

Méthode avancée avec `items`

Il peut être utile de parcourir à la fois sur les clés et sur les valeurs, c'est à dire sur l'élément complet `clé: valeur` du dictionnaire, notamment si on souhaite récupérer la clé associée à une valeur donnée.

Script Python

```
>>> for k, v in contacts.items():
...     print(v, "est associé à", k)
...
0516526600 est associé à bénédicte
0545974500 est associé à marc
0545387000 est associé à xavier
```

Création d'une liste vide et construction de dictionnaire

Comme les listes, il est très fréquent qu'on ait besoin de construire le dictionnaire par ajouts successifs en partant d'un dictionnaire vide.

On peut créer un dictionnaire vide de deux façons: `{}` ou `dict()`:

Script Python

```
>>> dico = {}
>>> dico = dict()
>>> chiffres = ['zéro', 'un', 'deux', 'trois', 'quatre', 'cinq', 'six', 'sept',
'huit', 'neuf']
>>> for c in chiffres:
...     dico[c] = len(c)
...
>>> dico
{'zéro': 4, 'un': 2, 'deux': 4, 'trois': 5, 'quatre': 6, 'cinq': 4, 'six': 3, 'sept':
4, 'huit': 4, 'neuf': 4}
```

On peut bien évidemment créer un dictionnaire en compréhension (**hors-programme**). On peut résumer le code précédent en:

Script Python

```
dico = {c: len(c) for c in chiffres}
```

3. 2.3.3 Exercices

Exercice 1

Énoncé

On considère le contenu de mon dressing, représenté par un dictionnaire:

Script Python

```
dressing = {"pantalons": 3, "pulls": 4, "tee-shirts": 8}
```

1. J'ai oublié mes 5 chemises... Ajoutez-les au dictionnaire `dressing`.
2. Écrire une fonction (ou plutôt une *procédure*, pas de **return**) `ajout(habit)` qui prend en paramètre un type d'habit et qui augmente son nombre de 1 dans le dictionnaire.
3. Améliorer la fonction pour qu'elle crée une nouvelle entrée si le type d'habit n'existe pas comme clé.

Correction

Exercice 2

Énoncé

On considère la liste suivante :

Script Python

```
lst = ['5717', '1133', '5545', '4031', '6398', '2734', '3070', '1346', '7849', '7288',
'7587', '6217', '8240', '5733', '6466', '7972', '7341', '6616', '5061', '2441',
'2571', '4496', '4831', '5395', '8584', '3033', '6266', '2452', '6909', '3021',
'5404', '3799', '5053', '8096', '2488', '8519', '6896', '7300', '5914', '7464',
'5068', '1386', '9898', '8313', '1072', '1441', '7333', '5691', '6987', '5255']
```

Quel est le **chiffre** qui revient le plus fréquemment dans cette liste ?

Correction

Exercice 3

Énoncé

On modélise des informations (nom, taille et poids) sur des Pokémons de la façon suivante:

Script Python

```
exemple_pokemons = {
    'Bulbizarre': (70, 7),
    'Herbizarre': (100, 13),
    'Abo': (200, 7),
    'Jungko': (170, 52)}
```

Par exemple, `Bulbizarre` est un pokémon qui mesure 70 cm et pèse 7 kg.

1. Ajouer le pokémon `Goupix` qui mesure 60 cm et pèse 10 kg.
2. Compléter la fonction `plus_grand` qui prend en paramètre un dictionnaire et qui renvoie un tuple contenant le nom du pokemon le plus grand et sa taille.

Script Python

```
def plus_grand(pokemons: dict) -> tuple:
    grand = ''
    taille_max = 0
    for  in pokemons. :
        if taille > taille_max:
            grand =
            taille_max =
    return
```

Correction

Exercice 4

Énoncé

Reprendre l'exercice «Le lion de Némée» et construire un dictionnaire dont les clés sont les noms des divinités et les valeurs leur «valeur» selon l'énoncé.

Correction

Exercice 5

Énoncé

Voici un dictionnaire `dates` dont les clés sont des prénoms au format `str` et les valeurs des dates de naissance au format `tuple`.

Script Python

```
dates = {"Alan": (23, 6, 1912),
         "Grace": (9, 12, 1906),
         "Linus": (28, 12, 1969),
         "Guido": (31, 1, 1956),
         "Ada": (10, 12, 1815),
         "Tim": (8, 6, 1955),
         "Dennis": (9, 9, 1941),
         "Hedy": (9, 11, 1914),
         "Steve": (24, 2, 1955)
        }
```

Par exemple, Linus est né le 28 décembre 1969.

1. Ajouter les deux entrées suivantes: Margaret, née le 17 août 1936 et John, né le 28 décembre 1903.
2. Écrire une fonction `calendrier` qui prend en paramètre un dictionnaire constitué d'entrées `nom: (jour, mois, année)` comme `dates` et qui renvoie un dictionnaire dont les clés sont les mois de l'année et les valeurs les listes des noms nés ce mois-là. Par exemple, `calendrier(dates)` doit renvoyer un dictionnaire contenant l'entrée `"juin": ["Alan", "Tim"]`.
3. Écrire une fonction `plus_jeune` qui renvoie le nom de la personne la plus jeune du dictionnaire.

Correction