1 Rappels : les types de base en Python

En Python, les données peuvent être stockées selon différents types de base (on parle aussi de **class**) qui sont : les entiers (int), les chaines de caractères (string), les flottants (float), les t-uplets (tuple) et les listes (list).
Dans ce thème, nous allons découvrir une nouvelle manière de stocker des données : les **dictionnaires** (dict).

2 Le type dictionnaire (dict) en Python

Le type DICT en Python

En Python, un objet de la classe dictionnaire est une **collection NON ORDONNÉE** (pas d'index) d'objets. L'ordre dans lequel apparaissent les objets n'a pas d'importance et n'est pas choisi par l'utilisateur.
Un dictionnaire est organisé sous le modèle associatif :

$$clé : valeur$$

Le symbole pour marquer le début et la fin de la collection est **l'accolade**.

Remarques : une clé est forcément unique et peut être de type : nombre, chaine de caractère, tuple. Une clé ne peut pas contenir une donnée mutable, donc pas de liste Python.

Exemples : Voici trois dictionnaires. On remarquera que les éléments d'un dictionnaire sont séparés par une virgule, comme dans un tableau.

```
nombres = { 0 : "zéro" , 1 : "un" , 2 : "deux"}

stock = {"crayons" : 43 , "gommes" : 18 , "feutres" : 5 }

plan = { (0,0) : "gare" , (10,8) : "mairie" }
```

3 Les outils de base des dictionnaires

1. On peut connaitre le nombre d'élément d'un dictionnaire avec la méthode **len** :
   `>>> len(nomDuDico)`
2. On peut accéder à la valeur associée à une clé :
   `>>> NomDuDico[clé]`
3. On peut afficher un dictionnaire :
   `>>> nomDuDico` ou `print(nomDuDico)`
4. on peut **supprimer** une clé et sa valeur associée :
   `>>> del(nomDuDico[clé])`
5. on peut **ajouter ou modifier** une clé :
   `>>> nomDuDico[clé] = maValeur`
   Si la clé existe, la valeur est modifiée, sinon elle est ajoutée.
6. on peut **tester** si une clé est présente ou pas dans dictionnaire :
   `>>> clé in nomDuDico` renvoie **True** si la clé est présente, et **False** sinon.
7. on peut créer un dictionnaire vide :
   `>>> nomDuDico = {}`

Remarque : on ne peut pas ajouter (+) ni multiplier (*) des dictionnaires. Nous allons voir plus tard comment **fusionner** deux dictionnaires.

Remarque : comme un dictionnaire n'est pas une séquence ordonnée, il n'y a pas de **slice** comme avec les listes Python. Par exemple `monDico[2 :5]` ne fonctionne pas...

4 Parcourir automatiquement un dictionnaire

parcourir un dictionnaire

Un dictionnaire est **itérable**, c'est à dire qu'il peut être parcouru avec la boucle FOR.

Exemple : la boucle ci-dessous parcourt le dictionnaire et affiche chaque clé et sa valeur associée. La variable **clé** prendra successivement la valeur ce chacune des clés présentes dans le dictionnaire.

```
for clé in nomDuDico :
        print(clé)
        print(nomDuDico[clé])
```

5 Les méthodes spéciales des dictionnaires en Python

Dans cette partie on parle des méthodes spéciales associées aux dictionnaires. Selon le paradigme de la programmation, ces méthodes sont accessibles avec la notation pointée `nomDuDico.methode()`.
Pour illustrer les méthodes spéciales, nous allons prendre le dictionnaire suivant :

```
stock = {"crayons" : 43 , "gommes" : 18 , "feutres" : 5 }

stock2 = {"crayons" : 8, "stylos" : 1 }
```

La méthode keys()

la méthode **keys()** permet de **récupérer sous forme d'itérable toutes les clés d'un dictionnaire**.
On peut transformer cet itérable en liste Python avec la commande **list**.

Exemple : tester le code ci-dessous.

```
liste_cle = stock.keys()
print(liste_cle)
l1 = list(liste_cle)
print(l1)
```

Vérifier le type de « liste_cle » et celui de « l1 ». Vous pouvez constater que `l1 = ['crayons', 'gommes', 'feutres']`.

La méthode values()

la méthode **values()** permet de **de récupérer sous forme de liste Python toutes les valeurs d'un dictionnaire**.

Exemple : tester le code ci-dessous.

```
liste_val = stock.values()
l2 = list(liste_val)

print(liste_val)
print(l2)
```

Vérifier le type de « liste_val » et celui de « l2 ». Vous pouvez constater que `l2 = [43, 18, 5]`.

La méthode items()

la méthode **items()** permet de récupérer sous forme de liste Python toutes les couples `clé : valeur` d'un dictionnaire sous forme d'une **liste de couples**.

Exemple : tester le code ci-dessous.

```
liste_items = stock.items()
l3 = list(liste_item)

print(liste_item)
print(l3)
```

Vérifier le type de « liste_items » et celui de « l3 ».
Vous pouvez constater que `l3 = [('crayons', 43), ('gommes', 18), ('feutres', 5)]`.

La méthode get()

la méthode **get(clé, valeur_par_défaut)** renvoie la valeur de la clé spécifiée **si la valeur est pas dans le dictionnaire**, et si la clé n'est pas dans le dictionnaire, elle renvoie le second argument.

Exemple : tester le code ci-dessous.

```
v1 = stock.get("crayons", None)
v2 = stock.get("billes", None)

print(v1)
print(v2)
```

Vous pouvez constater que `v1 = 43` car la clé "crayons" est bien dans le stock, alors que `v2 = None` car la clé "billes" n'est pas dans le stock. Utiliser la méthode `get()` est préférable à `stock[clé]` si on est pas sûr que la clé est dans le dictionnaire, car alors une erreur est renvoyée... Vérifier cela en saisissant `stock["billes"]`.

La méthode update()

la méthode **update(dico_2) met à jour** le contenu d'un dictionnaire avec celui passé en argument.

- Si la clé de **dico_2** n'est pas présente dans le dictionnaire, alors elle est ajoutée avec sa clé ;
- si la clé de **dico_2** est déjà présente, alors la valeur de associée est changée par celle de **dico_2**.

Exemple : tester le code ci-dessous.

```
stock.update(stock2)

print(stock)
```

La fonction **update()** ne renvoie rien : elle se contente de mettre à jour le dictionnaire **stock** avec celui de **stock2**. On remarque le changement de valeur pour "crayons", l'ajout de la clé "stylos" avec sa valeur associée.
Vous pouvez constater que stock = {'crayons' : 8, 'gommes' : 18, 'feutres' : 5, 'stylos' : 1} , et aussi que le contenu de "stock2" n'est pas changé.

La méthode clear()

la méthode **clear()** permet de **vider un dictionnaire**.

Exemple : la commande stock.clear() vide le dictionnaire « stock ». Tester cette commande en ligne de commande...

6 La création de dictionnaires et de listes Python par compréhension

Il est possible de créer un dictionnaire **par compréhension**, un peu de la même manière qu'on le fait pour des listes.

Exemple 1 : tester les quelques exemples de création de listes Python par compréhension, et analyser le résultat.

```
liste1 = [0 for n in range(5)]

liste2 = [ 2*n for n in range(10) ]

liste3 = [lettre for lettre in 'mot']
```

Remarque : la **liste1** peut aussi être obtenue avec la commande liste1 = [0]*5

Exemple 2 : dans cet exemple, nous montrons comment créer un dictionnaire par compréhension à partir d'un liste. Programmez et observez le résultat obtenu.

```
liste = ["France", "Espagne", "Italie"]

dic1 = { pays : 0 for pays in liste }

couples = [ ("a", 1) , ("b", 2), ("c",3) ]

dic2 = { lettre : rang for (lettre, rang) in couples }
```

Remarque : vous devez observer que dic1 = {'France' : 0, 'Espagne' : 0, 'Italie' : 0}

Application : on donne une liste de mots. Créer par compréhension un dictionnaire qui à chaque mot associe son nombre de lettres.

```
mots = ["bonjour", "tout", "le", "monde"]

dic3 = ...
```

Au final, on doit avoir dic3 = {'bonjour' : 7, 'tout' : 4, 'le' : 2, 'monde' : 5}