- Représentation des données : types construits -

LES DICTIONNAIRES

Plan du chapitre

I. INTRODUCTION

Les types de données composites que nous avons abordés jusqu'à présent (chaines, listes et tuples) étaient tous des *séquences*, c'est-à-dire une suite ordonnée d'éléments auxquels on peut accéder à la condition de connaitre leur emplacement (identifié par leur indice).

Les ***dictionnaires***, objet de ce chapitre, constituent eux aussi un type de *données composites*, mais ne sont pas des *séquences* dans le sens où les éléments qui les constituent ne sont pas disposés dans un ordre immuable. Les éléments sont repérés non pas par des indices ordonnés, mais par une **clé**, laquelle pouvant être de type quelconque (str, int…) à laquelle on va attribuer une valeur.

Exemple d'un dictionnaire de langues :

clé	valeur
"yes"	"oui"
"no"	"non"
"or"	"ou"
"and"	"et"

Pour créer ce dictionnaire, on écrit entre accolades les couples séparés par des virgules, chaque couple étant composé d'une clé et de la valeur correspondante séparés par deux- points :

```
1 dico={"yes":"oui", "no":"non", "or":"ou",  "and":"et"}
2 print(dico)
```

Comme pour les chaines de caractères, les tuples ou les listes, la fonction len() retourne la longueur du dictionnaire, c'est-à-dire le nombre de paires clé-valeur :

```
1 dico={"yes":"oui", "no":"non", "or":"ou", "and":"et"}
2 print (len(dico))
```

Retour au plan

II. ACCÈS AUX ÉLÉMENTS DU DICTIONNAIRE

Deux méthodes nommées keys() et values() donnent accès aux clés et aux valeurs.
La méthode items() donne accès à l'ensemble des couples clé-valeur :

```
1 dico={"yes":"oui", "no":"non", "or":"ou",  "and":"et"}
2
3 print(dico.keys())
4 print(dico.values())
5 print(dico.items())
```

Pour accéder à une valeur particulière du dictionnaire on travaille comme avec une liste, mais on précise la clé à la place de l'indice :

```
1 dico={"yes":"oui", "no":"non", "or":"ou",  "and":"et"}
2
3 print(dico["yes"])
```

Attention, si l'on demande une clé qui n'existe pas, alors une erreur arrête le programme :

```
1 dico={"yes":"oui", "no":"non", "or":"ou", "and":"et"}
2
3 print (dico["if"])
```

Pour remédier à ce problème, on utilise la méthode get qui renvoie la valeur si elle existe, sinon renvoie None :

```
1 dico={"yes":"oui", "no":"non", "or":"ou", "and":"et"}
2
3 print (dico.get("if"))
```

Retour au plan

III. AJOUT OU SUPPRESSION D'ÉLÉMENTS

Pour ajouter des éléments à un dictionnaire, il n'est pas nécessaire de faire appel à la méthode append() utilisée [2]pour les listes. Il suffit simplement de créer une nouvelle paire clé-valeur.

```
1 inventaire={"pommes": 50, "oranges":30, "abricots": 80}
2
3 # ajout de poires à l'inventaire
4 inventaire["poires"]=25
5 print(inventaire)
```

Pour supprimer un élément à un dictionnaire, on utilise l'instruction del.

```
inventaire={"pommes": 50, "oranges":30, "abricots": 80}

# suppression des oranges de l'inventaire
del inventaire["oranges"]
print(inventaire)
```

Retour au plan

IV. TEST D'APPARTENANCE

Tout comme avec les chaines, les tuples et les listes, l'instruction in est utilisable avec les dictionnaires

```
inventaire={"pommes": 50, "oranges":30, "abricots": 80}

if "oranges" in inventaire :
    print ("il nous reste ", inventaire["oranges"], " oranges en stock")
else :
    print("il ne reste plus d'oranges")
```

Retour au plan

V. PARCOURS D'UN DICTIONNAIRE

Avec une boucle for, il est possible d'itérer sur les clés. L'ordre dans lequel les éléments seront extraits est imprévisible puisqu'un dictionnaire n'est pas une séquence.

```
inventaire={"pommes": 50, "oranges":30, "abricots": 80}

for i in inventaire :
    print("la clé ", i ," a pour valeur ", inventaire[i])
```

Il est aussi possible d'itérer sur les items :

```
inventaire={"pommes": 50, "oranges":30, "abricots": 80}

for i,j in inventaire.items() :
    print("la clé ", i ," a pour valeur ", j)
```

La méthode items appliquée au dictionnaire inventaire renvoie une séquence de tuples (clé, valeur)

Retour au plan

VI. COMPARAISON DE DICTIONNAIRES

Comme nous l'avons déjà dit, les dictionnaires ne sont pas des séquences, la place des éléments dans le dictionnaire est imprévisible, ce qui rend problématique la comparaison de deux dictionnaires.

L'opérateur & permet de comparer les clés de 2 (ou plusieurs) dictionnaires :

```
inventaire1={"pommes": 50, "oranges":30, "abricots": 80}
inventaire2={"pommes": 30, "oranges":18, "abricots": 80, "bananes":10}

print (inventaire1.keys() & inventaire2.keys())
```

Il permet aussi de comparer non seulement les clés mais aussi les couples clé-valeur :

```
inventaire1={"pommes": 50, "oranges":30, "abricots": 80}
inventaire2={"pommes": 30, "oranges":18, "abricots": 80, "bananes":10}

print (inventaire1.items() & inventaire2.items())
```

Pour connaitre les clés qui figurent dans *inventaire2* mais pas dans *inventaire1*, on procède ainsi :

```
inventaire1={"pommes": 50, "oranges":30, "abricots": 80}
inventaire2={"pommes": 30, "oranges":18, "abricots": 80, "bananes":10}

print (inventaire2.keys() - inventaire1.keys())
```

Retour au plan

VII. COPIE D'UN DICTIONNAIRE

Avec les dictionnaires on rencontre le même problème qu'avec les listes :

```
inventaire1={"pommes": 30, "oranges":18, "abricots": 80}

inventaire2 = inventaire1
inventaire2["pommes"]=100

print (inventaire1)
print (inventaire2)
```

On pourrait penser que le nombre de pommes est toujours de 30 dans l'inventaire1 et de 100 dans l'inventaire2. Il n'en est rien puisque l'affichage montre que l'inventaire1 a également été modifié. En fait inventaire1 et inventaire2 pointent vers le même emplacement mémoire (on peut considérer que l'on a deux noms pour le même dictionnaire).

Pour remédier à ce problème on utilise la fonction dict() qui renvoie un dictionnaire :

```
inventaire1={"pommes": 30, "oranges":18, "abricots": 80}

inventaire2 = dict(inventaire1)
inventaire2["pommes"]=100

print (inventaire1)
print (inventaire2)
```

Il est possible d'utiliser également la méthode copy() :

```
inventaire1={"pommes": 30, "oranges":18, "abricots": 80}

inventaire2 = inventaire1.copy()
inventaire2["pommes"]=100

print (inventaire1)
print (inventaire2)
```

Remarque : la fonction dict() peut également être appliquée à une liste de tuples afin de créer un dictionnaire :

```
listTuples = [("pommes", 30), ("oranges",18), ("abricots", 80)]

inventaire = dict(listTuples)
print (inventaire)
```

Retour au plan

VIII. DES DICTIONNAIRES DE DICTIONNAIRES

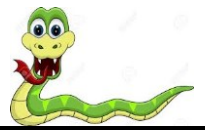
De même que les éléments d'une liste peuvent être des listes, les éléments d'un dictionnaire peuvent être des dictionnaires :

```
pays = {"france":{"capitale":"Paris", "population":6780000, "superficie":643800},
        "portugal":{"capitale":"Lisbonne", "population":10300000, "superficie":92300}
        }

print(pays.get('france'))
print(pays.get('france').get('capitale'))
```

Retour au plan

IX. PROJET : MÉTADONNÉES EXIF



Les fichiers image au format jpg contiennent plus que la simple image, on y trouve aussi des informations sur l'image elle-même (définition, résolution, profondeur de couleur, compression...), sur la prise de vue (date et heure), sur le lieu de la prise de vue (coordonnées GPS), sur l'appareil et la totalité de ses réglages (marque, modèle, temps exposition, focale…). Ces données contenues dans un fichier image s'appellent des métadonnées EXIF (EXchangeable Image file Format). Il est possible de visualiser les principales depuis l'explorateur Windows en effectuant un clic droit sur l'image, puis en sélectionnant propriétés et détails.

La bibliothèque PIL (Python Image Librairy) comporte une méthode nommée _getexif() qui retourne ces métadonnées sous forme d'un dictionnaire :

```
1 from PIL import Image
2
3 img=Image.open("photo.jpg")  # la photo nomée photo.jpeg est dans le répertoire du script
4 dico=img._getexif()
5 print(dico)
```

Après avoir exécuté le programme, utiliser l'explorateur de variables" de Spyder, pour analyser le contenu du dictionnaire. Pour voir la totalité du dictionnaire relativement complexe, double-cliquer sur valeur du dictionnaire afin de visualiser les paires clé-valeur.

Nous voyons que les clés sont des nombres entiers. On comprend rapidement à partir de l'explorateur de variables que par exemple la valeur correspondant à la clé 271 n'est autre que la marque de l'appareil (appareil photo, scanner…) qui a permis de numériser l'image. Le site http://www.exiv2.org/tags.html donne la signification de l'ensemble des clés.

Explication concernant les coordonnées GPS du lieu où a été prise la photo :
Les coordonnées GPS sont la valeur de l'élément du dictionnaire dico dont la clé est 34853. Elles se présentent sous forme d'un 2nd dictionnaire dont :

- La clé 1 donne l'hémisphère : N ou S.
- La clé 2 donne 3 tuples qui correspondent à la latitude exprimée en degrés, minute, seconde (DMS).
  Si les " tuples sont (47,1), (37,1), (29107360, 1000000), il faut comprendre 47/1 degrés 37/1 minutes et 29107360/1000000 = 29,107360 secondes, aussi noté 47° 37' 29,107360"
  La latitude en DMS s'écrira donc sous la forme 47° 37' 29.10736" N
- La clé 3 précise l'est (E) ou l'ouest (W) par rapport au méridien de Greenwich.
- La clé 4 donne également 3 tuples qui correspondent à la longitude en DMS (comme la clé 2).
  La longitude en DMS s'écrira donc sous la forme 3° 25' 42.97657" W

Certains logiciels de cartographie utilisent les degrés décimaux (DD). Afin de convertir des DMS en DD il faut appliquer la formule suivante : DD = degrés+(minute/60) +(seconde/3600) et ajouter un signe moins devant la longitude au format DD si nous sommes situés à l'ouest du méridien de Greenwich et un signe moins devant la latitude si nous sommes dans l'hémisphère sud (N,S,E,W n'ont donc plus à être précisés)

TRAVAIL DEMANDÉ : réaliser un logiciel qui à sera capable d'extraire les métadonnées Exif d'un fichier jpg et de créer un nouveau dictionnaire nommé *meta* avec les clés et valeurs du tableau ci-dessous :

Clé	Valeur (type)
"date"	Date et heure de la prise de vue (str)
"largeur"	Largeur de l'image en pixels (int)
"hauteur"	Hauteur de l'image en pixels (int)
"couleur"	Profondeur de couleur en bits (int)
"marque"	Marque de l'appareil photo (str)
"modele"	Modèle de l'appareil photo (str)
	Coordonnées GPS :
"latitudeDMS"	Latitude en DMS sous la forme 47.0° 37.0' 29.10736" N (str)
"latitudeDD"	Latitude en DD sous la forme 47.62475204444444 (float)
"longitudeDMS"	Longitude en DMS sous la forme 3.0° 25.0' 42.97657" W (str)
"longitudeDD"	Longitude en DD sous la forme -3.4286046027777775 (float)
"altitude"	Altitude en m (float)

Retour au plan