CH 6 : LES TYPES DE DONNEES CONSTRUITS

Nous avons déjà étudié un type de données construits : les listes.

Nous allons étudier dans ce chapitre deux autres types : les p-uplets (ou tuple) et les dictionnaires.

I. LES P-UPLETS (ou tuple)

1. Définition

A retenir

Un p-uplet est un objet de type tuple. C'est une suite ordonnée d'éléments qui peuvent être de n'importe quel type, y compris de types différents (ce qui n'est pas recommandé pour une liste).

Un p-uplet s'écrit avec p valeurs séparées par des virgules :

```
>>> t=1 , 2 , 3 , 'bonjour'
>>> t
(1, 2, 3, 'bonjour')
>>> type(t)
<class 'tuple'>
```

Pour créer un p-uplet vide, on écrit :

```
>>> t=()
```

Pour créer un p-uplet à un seul élément, on écrit (la virgule est obligatoire pour spécifier que l'on crée un p-uplet) :

```
>>> t=(3 ,)
```

Un tuple peut être constitué de tuples :

```
>>> t=(1 , 2 , 3 ,4 , 'bonjour')
>>> u=(t , (4 , 5 , 6 , 7, 'au revoir'))
>>> u
((1, 2, 3, 4, 'bonjour'), (4, 5, 6, 7, 'au revoir'))
>>> len(u)
2
```

2. Utilisation

Comme pour les listes, les éléments d'un tuple sont numérotés à partir de 0, et la syntaxe pour accéder à un élément particulier est la même.

```
>>> t[0]
1
>>> u[0]
(1, 2, 3, 4, 'bonjour')
>>> u[0][0]
1
```

→ Quelle est la syntaxe pour accéder au '4' ?

A retenir :

Comme les chaînes de caractères et contrairement aux listes, les tuples ne sont pas modifiables par une affectation de la forme t[i] = ... On dit que ce sont des objets *IMMUTABLES*.

On ne peut pas non plus les modifier avec les méthodes tu type t.append(). On ne peut donc pas modifier un tuple « en place », il faut créer un nouveau tuple.

Les opérateurs + et * s'utilisent comme pour les chaînes de caractères ou les listes.

```
>>> t1 = (1 , 2 , 3)
>>> t2 = (4 , 5 , 6)
>>> t=t1+t2
>>> u=2*t1
>>> t
(1, 2, 3, 4, 5, 6)
>>> u
(1, 2, 3, 1, 2, 3)
```

On peut tester l'appartenance d'un élément à un tuple grâce à l'instruction in :

```
>>> 1 in t
True
>>> 7 in u
False
```

Application

TP jeu de Uno

II. LES DICTIONNAIRES

1. Définition

Les éléments d'une liste sont repérés par des indices 0, 1, 2, ... Une différence essentielle avec un dictionnaire, objet de type dict, est que les indices sont remplacés par des objets du type str, float, tuple (si les n_uplets ne contiennent que des entiers, des flottants ou des p-uplets). On les appelle des clés et à chaque clé correspond une valeur. Ces clés ne sont pas ordonnées.

A retenir

Les éléments d'un dictionnaire sont des **couples clé-valeur**. Un dictionnaire est créé avec des accolades, les différents couples étant séparés par des virgules. La clé et la valeur correspondante d'un élément sont séparées par deux points.

```
>>> dico = {"A" : 0 , "B" : 1 , "C" : 2}
>>> type(dico)
<class 'dict'>
>>> dico
{'A': 0, 'B': 1, 'C': 2}
```

On peut aussi construire un dictionnaire en compréhension comme avec les listes (tester et vérifier la valeur de d) :

```
>>> d = {chr(65+i) : i for i in range(26)}
```

On peut également convertir une liste de listes à deux éléments avec la méthode dict() :

```
>>> liste = [['a' , 0] , ['b' , 1] , ['c' , 2]]
>>> dico = dict(liste)
>>> dico
{'a': 0, 'b': 1, 'c': 2}
```

2. Utilisation

Pour accéder aux clés ou aux valeurs, nous avons les méthodes keys() et values().

```
>>> dico.keys()
dict_keys(['c', 'a', 'b'])
>>> dico.values()
dict_values([2, 0, 1])
```

Comme vous pouvez le constater, les clés et valeurs ne sont pas ordonnées en elles-mêmes, il y a juste une correspondance entre la clé et sa valeur.

Pour l'accès à l'ensemble des couples clés-valeurs, nous utilisons la méthode items().

```
>>> dico.items()
dict_items([('c', 2), ('a', 0), ('b', 1)])
```

Remarque : les couples clés-valeurs obtenus sont du type tuple.

On peut tester l'appartenance à un dictionnaire avec le mot clé in.

```
>>> 'a' in dico
True
>>> 3 in dico.values()
False
>>> ('c' , 2) in d.items()
False
```

On en déduit trois manières de parcourir un dictionnaire. Il est possible d'utiliser les clés, les valeurs, ou les couples clés-valeurs. Voici un exemple avec les clés :

```
>>> for key in dico:
...     print(key)
...
c
a
b
```

A retenir

L'accès à une valeur particulière s'obtient en précisant la clé.

Par exemple, dico["A"] a la valeur 0.

Il est possible de **modifier** une valeur par affectation comme dico["A"] = 1. Et l'instruction dico["D"] = 3 ne provoque pas d'erreur : une **nouvelle clé est créée**.

La méthode len() renvoie le nombre d'éléments d'un dictionnaire, sa longueur.

Pour supprimer un élément, on utilise l'instruction del dico[clé] .

```
>>> del dico['a']
>>> dico
{'b': 1, 'c': 2}
```

3. Applications aux images numériques

Types d'image

On désigne sous le terme d'image numérique toute image acquise, créée, traitée ou stockée sous forme binaire (suite de 0 et de 1).

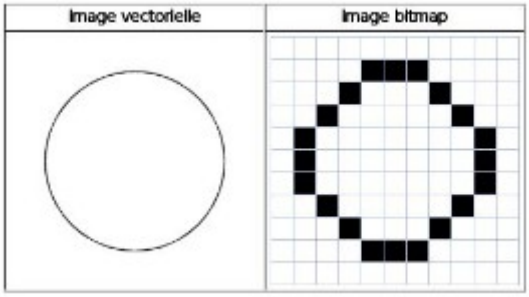
On distingue 2 familles :

- **Les images matricielles**

(.bmp .png .jpeg .tiff .gif … etc)

Elles sont composées d'une matrice (tableau 2D) de points colorés. Dans le cas des images à deux dimensions (le plus courant), les points sont appelés pixels.

Des exemples ici :
https://www.lestutosdepepin.fr/format_images.K.htm



(https://www.apprendre-en-ligne.net/info/images/images.pdf)

- **Les images vectorielles**

(.svg .ps .vml … etc)

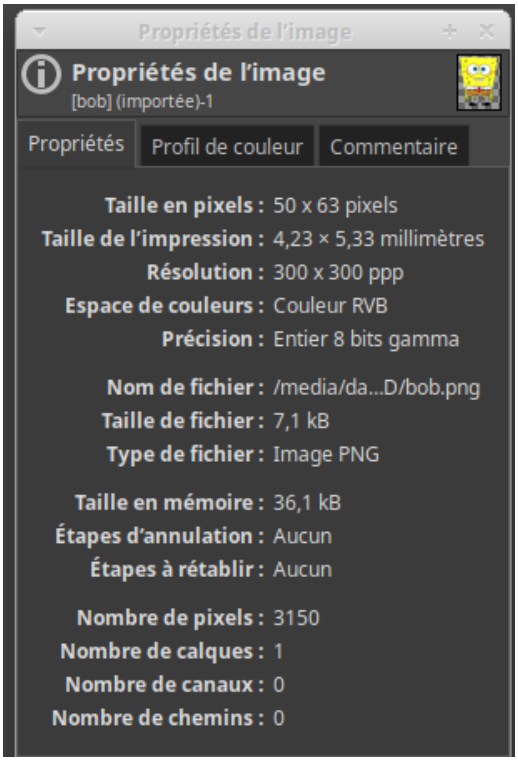
Les images vectorielles est de représentent les données de l'image par des formules géométriques qui vont pouvoir être décrites d'un point de vue mathématique. Cela signifie qu'au lieu de mémoriser une mosaïque de points élémentaires, on stocke la succession d'opérations conduisant au tracé.
Un intérêt de l'image vectorielle est qu'on peut la redimensionner sans perte de qualité.

Dans ce chapitre, nous n'aborderons que les images de type matricielles.

(image wikipédia)

Méta-données

Dans un éditeur d'image comme Gimp, on peut visualiser les propriétés de l'image affichée.



Par exemple,

Taille :

Largeur x Hauteur

Elle donne les dimensions de l'image en pixels.

Résolution :

Résolution Largeur x Résolution Hauteur

Elle donne la qualité de l'image en ppp (*Pixels Par Pouce*), c'est à dire la densité de pixels par unité de longueur.

Ces 2 propriétés permettent d'avoir une idée de l'espace mémoire nécessaire au stockage, et la taille optimale d'affichage.

Ces propriétés et beaucoup d'autres informations sont contenues dans les méta-données EXIF de l'image (*EXchangeable Image File format*) sous forme de dictionnaire.
Les spécifications sont gérées par un organisme japonais, le JEITA, qui définit différents dictionnaires de référence permettant d'accéder à ces données. Les clés (les tags) et les valeurs sont des nombres écrits dans l'entête du fichier. Les dictionnaires donnent l'interprétation de ces clés.

Par exemple la clé 256 a pour valeur « Width », la largeur de l'image en pixel, la clé 257 a pour valeur « Length », la hauteur de l'image en pixel.

Il est possible de voir les données EXIF complètes avec python ou des logiciels spécifiques (http://exif.regex.info/exif.cgi)

Applications :

TP Traitement d'image

MINIPROJET5 Création d'un logiciel (simplifié) de traitement d'image