IngeSUP - Cours 10 - Structures de données avancées I

Sommaire

Fonctionnement et règles

- Rappel sur les listes
- Créer et manipuler une liste de listes
- Créer des structures de données de type `tuple`, `dict` et `set` ;
- Utilisation de ces stuctures de données;

Introduction

Lors de la séance 4 nous avons vu les listes en Python. Ls listes permettent de ranger dans un ordre donné, des éléments de même types ou non, d'ajouter ou d'enlever des éléments, de les trier etc. C'est la **structure de données** la plus utilisé dans les algorithmes simples.

Une **structure de donnée** est un objet dans un programme qui contient une série de données. Une structure de données simple peut être par exemple une liste qui contient les composantes d'un vecteur ou une liste de noms. Une structure de données plus complexe serait par exemple un répertoire téléphonique contenant les associations entre les numéros de téléphone et les noms.

Par exemple, si nous souhaitons stocker des noms d'étudiants par groupe de labo, au lieu d'utiliser des variables de type chaîne de caractères pour chaque étudiant, nous pouvons utiliser une liste de chaînes de caractère :

Entrée [ ]:

```
lab_group0 = ["Théo", "Emilie", "Sarah", "Marc"]
```

Cette construction est plus puissante car nous pouvons effectuer de nombreuses opérations sur les listes, comme vérifier leur longueur (le nombre d'étudiants dans un groupe), trier les noms par ordre alphabétique, ajouter/supprimer des noms. Nous pouvons même faire des listes de listes.

En Python, nous pouvouns identifier quatre structures de données couramment utilisées :

- les listes ;
- les tuples ;
- les dictionnaires;
- les ensembles.

Rappel sur les listes

Une liste peut être composé d'éléments de différents types `int`, `float` ou `str`. Il est également possible d'y ranger des listes de taille différentes.

Imaginons que nous voulions lister le nom et la promotion des membres des labos, nous pourrions utiliser une liste pour modéliser un membre dont le premier élément serait le prénom et le second serait sa promotion :

Entrée [ ]:

```
membre1 = ["Jean","Inge1"]
membre2 = ["Marie","IngeSup"]
print("prénom :",membre1[0])
print("promo  :",membre1[1])
print("prénom :",membre2[0])
print("promo  :",membre2[1])
```

Créer et manipuler une liste de liste

Si l'on souhaite maintenant construire la liste de tous les membres, il nous suffit d'ajouter toutes les listes membreX à une seule même liste **membres**. On a alors ce que l'on appelle une **liste de listes**. Dans ce cas, on a affaire à un objet à deux dimensions.

Entrée [ ]:

```
membres=[]
membres.append(membre1)
membres.append(membre2)
print(membres)
```

Le premier élément est alors une liste correspondant au premier membre de **membres**. On y accède par son indice **0**.

Entrée [ ]:

```
print(membres[0])
print(membres[1])
```

On peut alors boucler sur chaque éléments de **membres**.

Entrée [ ]:

```
for membre in membres:
    print(membre)
```

On peut aussi boucler sur chaque indice du tableau **membres**.

Entrée [ ]:

```
for i in range(len(membres)):
    print(i)
```

Si à partir des indices je veux accéder aux éléments , je fais :

Entrée [ ]:

```
for i in range(len(membres)):
    print(membres[i])
```

> **Information**
>
> **Rappel:**
>
> Utilisez `for variable in liste` pour boucler directement sur les valeurs de la liste
>
> Utilisez `for variable in range(len(liste))` si dans votre boucle les indices ont une importance

> **Note**
>
> Notez la convention "grammaticale" pour le choix des noms des variables. La liste (de listes) **membres** est au pluriel et la variable **membre** qui itére est au singulier. Cette simple convention permet une plus grande lisibilité du code.

Si l'on souhaite accéder au prénom (le premier élément de la liste **membre**) du premier élément de la liste **membres** directement, il faudra utiliser la construction à deux dimensions des crochets. Par exemple :

Entrée [ ]:

```
print(membres[0][0])
print(membres[1][0])
```

Ainsi il est possible d'accéder à l'ensemble des données d'une liste de liste par l'intermédiaire de **deux boucles imbriquées**.

Entrée [ ]:

```
for i in range(len(membres)):
    for j in range(len(membres[i])):
        print(i,j,membres[i][j])
```

Tuples (<class 'tuple'>)

Python propose un type de données appelé tuple (anglicisme informatique signifiant "Table UPLEt"), qui est assez semblable à une liste mais qui n'est pas modifiable (on dit qu'il est ***immutable***).

Un tuple est défini par des parenthèses.

Entrée [ ]:

```
tup = ("a","b","c")
print(type(tup))
print(tup)
```

Comme les listes, un tuple est **itérable** :

Entrée [ ]:

```
for ele in tup:
    print(ele)
```

Comme les listes, un tuple est **indéxable** (scriptable) :

Entrée [ ]:

```
print(tup[0])
print(tup[0:2])
```

Les tuples sont non **modifiables**.

Entrée [ ]:

```
liste = ["one", "two", "three ",2,3,4]
tup = ("one", "two", "three ",2,3,4)
print(tup)
print(liste)
```

Entrée [ ]:

```
liste[0] = "uno"
print(liste)
tup[0] = "uno"   # J'ai pas le droit !
print(tup)
```

Dictionnaires (<class 'dict'>)

Un dictionnaire en Python est une collection d'éléments non ordonnés. Alors que d'autres types de données composées ont uniquement la valeur en tant qu'élément, un dictionnaire possède une paire clé:valeur.

Un dictionnaire en Python fonctionne de manière similaire au dictionnaire dans un monde réel. Les clés d'un dictionnaire doivent être uniques et de type de données **immuable**, telles que chaînes, entiers et tuples, mais les valeurs-clés peuvent être répétées et être de n'importe quel type.

Chaque clé correspond à une valeur, nous ne pouvons donc pas avoir de clés dupliquées. Les dictionnaires sont des objets modifiables, ce qui signifie que nous pouvons ajouter, supprimer ou mettre à jour les éléments après leur création.

Les dictionnaires sont optimisés pour récupérer des valeurs lorsque la clé est connue.

Créer un dictionnaire

Créer un dictionnaire est aussi simple que de placer des éléments entre accolades {}, séparés par une virgule.

Entrée [ ]:

```
# dictionnaire vide
dict1={}
print("dict1 : ",dict1)
print(type(dict1))

#dictionnaire avec des clés de type chaines
dict2={"Prenom":"Boris", "age":20,"ville":"Moscou"}
print("dict2 : ",dict2)

# dictionnaire avec des clés entières
dict3= {1:"Meknes", 2:"Marrakech", 3:"Essaouira"}
print("dict3 : ",dict3)

# dictionnaire avec des clés mixtes
dict4={1:"Yoan", "ville":"Oslo", 10.4: 1.78}
print("dict4 : ",dict4)

# Création d'un dictionnaire avec la méthode dict ()
dict5 = dict({1: 'Dev', 2: 'Info', 3:'COM'})
print("dict5 : ",dict5)

# Créer un dictionnaire avec chaque élément en paire
dict6 = dict([(1, 'Geeks'), (2, 'For')])
print("dict6 : ",dict6)
```

Comme vous pouvez le voir ci-dessus, nous pouvons également créer un dictionnaire en utilisant la fonction intégrée `dict()` .

Comment accéder aux valeurs d'un dictionnaire ?

Comme indiqué précédemment, l'ordre des éléments dans un dictionnaire peut varier. Par conséquent, nous ne pouvons pas utiliser l'indice de l'élément pour accéder à la valeur. Au lieu de cela, nous utilisons une clé. Pour accéder à la valeur du dictionnaire, nous utilisons la syntaxe suivante:

```
nom_dictionnaire[clé] = valeur
```

Entrée [ ]:

```
D={1:"Thomas", "ville":"Paris", 10.4: 1.78}
print("Clé 1 a pour valeur : ", D[1])
print("clé ville a pour valeur ", D["ville"])
print("Clé 10.4 a pour valeur : ", D[10.4])
```

Si la clé spécifiée n'existe pas, une exception **KeyError** sera déclenchée.

Ajouter/Modifier des éléments

On ajoute un élément par l'intermédiaire de la clé entre crochets.

Entrée [ ]:

```
dic = {}
dic["pseudo"]="fifou"
dic["password"]="1234"
dic["nom"]="Philippe"
dic["bureau"]="203"
dic[2] = "oui"
print(dic)
```

Parcourir les clés, les valeurs, les éléments

Un dictionnaire est **itérable** par l'intermédiaire de la clé et de sa valeur à l'aide de la méthode **items** du dictionnaire.

Entrée [ ]:

```
for key,value in dic.items():
    print(key,value,dic[key])
```

Il est possible de boucler sur les clés ou les valeurs.

Entrée [ ]:

```
for key in dic.keys():
    print(key)
```

Entrée [ ]:

```
for values in dic.values():
    print(values)
```

Supprimer des éléments

Nous pouvons supprimer un élément particulier dans un dictionnaire en utilisant la méthode `pop()` . Cette méthode supprime comme élément avec la clé fournie et retourne la valeur.

La méthode `popitem()` peut être utilisée pour supprimer et renvoyer un élément arbitraire (clé, valeur) du dictionnaire. Tous les éléments peuvent être supprimés à la fois en utilisant la méthode `clear()` .

Nous pouvons également utiliser le mot-clé **del** pour supprimer des éléments individuels ou le dictionnaire entier lui-même.

Entrée [ ]:

```
D={1:"Adam", "ville":"Zurich", 10.4: 1.78, "age":32, 3:45, "tt":"test"}
print(D)

print("val : ", D.pop(1))
print("pop : ",D)

print("val : ", D.popitem())
print("popitem : ",D)

del D["age"]
print("del : ", D)

# vider le dictionnaire
D.clear()
print(D)
```

Ensemble (<class 'set'>)

Création d'un ensemble

Un ensemble est créé en plaçant tous les éléments (éléments) entre accolades {}, séparés par une virgule ou en utilisant la fonction intégrée `set()` .

Il peut avoir n'importe quel nombre d'éléments et ils peuvent être de types différents (int, float, tuple, string, etc.). Mais un ensemble **ne peut pas contenir d'éléments mutable**, comme une liste, un ensemble ou un dictionnaire.

Entrée [ ]:

```
# création d'un ensemble de 4 éléments
e1={1,3,5,7}
print("le type de e1 est : ",type(e1))
print("les éléments de e1 sont : ",e1)

e2={"Mostafa", 1.78, 32 }
print("e2 : ",e2)

# ensemble avec doublons
e3 = {3, 5, 7, 2, 3, 5}
print("e3 : ", e3)
```

> **Les ensembles ne peuvent pas avoir de doublons**

Nous pouvons également utiliser la fonction intégrée set() pour créer des ensembles. Voici quelques exemples:

Entrée [ ]:

```
# créer un ensemble à partir d'un ensemble
e1 = set({77, 23, 91, 271})
print("e1 : ",e1)

#créer un ensemble à partir d'une chaîne
e2 = set("123abc")
print("e2 : ",e2)

# créer un ensemble à partir d'une liste
e3 = set(['Valery', 'Giscard', "D'Estaing", 32, 1.78])
print("e3 : ",e3)

#créer un ensemble à partir d'un tuple
e4 = set(("Mehdi", "Marrakech", 18))
print("e4 : ",e4)
```

> **Information**
>
> Comme les ensembles en python ne peuvent pas avoir de doublons, il existe une astuce pour **supprimer tous les doublons présents dans une liste**. Il suffit de transformer cette liste en ensemble grâce à `set` puis de retransformer l'ensemble débarassé des doublons en liste.
>
> Voici un exemple d'utilisation :
>
> ```
> test_list = [1, 5, 3, 6, 3, 5, 6, 1]
> print("Verson originale : ", test_list)
>
> test_list = list(set(test_list))
> print("Version sans les duplicata : ", test_list)
> ```

Modifier un ensemble

Les ensembles sont **mutables**. Mais comme ils ne sont pas ordonnés, l'indexation n'a pas de sens. Nous ne pouvons pas accéder à un élément d'un ensemble ni le modifier à l'aide de l'indexation ou du découpage en tranches. **Set** ne le supporte pas.

Nous pouvons ajouter un seul élément à l'aide de la méthode `add()` et plusieurs éléments à l'aide de la méthode `update()`. La méthode `update()` peut prendre pour argument des tuples, des listes, des chaînes ou d'autres ensembles. Dans tous les cas :

- les doublons sont évités.
- Les tuples, listes, chaines ajoutés sont éclatés.

Entrée [ ]:

```
e={1,3}

# ajouter un élément
e.add(5)
print("e : ",e)

# ajouter une liste
e.update([9,4,7])
print("e : ",e)

# ajouter une liste et un ensemble
e.update([50,51],{100,200})
print("e : ",e)
```

Supprimer un élément d'un ensemble

Un élément particulier peut être supprimé de la série à l'aide des méthodes, `discard()` et `remove()`. La seule différence entre les deux est que, tout en utilisant `discard()` si l'élément n'existe pas dans l'ensemble, il reste inchangé. Mais si vous supprimez un élément qui n'existe pas à l'aide de `remove()` ça provoquera une erreur (**KeyError**).

Entrée [ ]:

```
e = {1, 3, 6, 7}

e.discard(3)
print("e : ", e)

e.remove(7)
print("e : ", e)

e.discard(9)
print("e : ", e)

e.remove(9)
print("e : ", e)
```

De même, nous pouvons supprimer et retourner un élément en utilisant la méthode `pop()`. Les ensembles n'étant pas ordonnés, il n'y a aucun moyen de déterminer quel élément sera supprimé. **C'est complètement arbitraire**.

Nous pouvons également supprimer tous les éléments d'un ensemble en utilisant `clear()`.

Entrée [ ]:

```
e={2, 4, 90, 100, 30, 32, 1}
val=e.pop()
print("val : ",val)
print("e : ",e)

e.clear()
print("e : ",e)
```

Opérations ensemblistes

Entrée [ ]:

```
A = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
B = {1, 2, 3, "one", "two", "three"}
print(A)
print(B)
print("")
print("Intersection: ",A & B) #intersection
print("Union: ",A | B) #union
print("Différence: ",B - A) #différence
print("Union disjointe: ",A ^ B) #union disjointe
```

Il est possible de tester la présence d'un élément d'un ensemble.

Entrée [ ]:

```
print(36 in A)
print("one" in B)
print(3 not in B)
```

Exercices de TD

Vous pouvez maintenant vous exercer à partir du notebook TD 10 (../TD/TD%2010%20-%20Les%20structures%20de%20donn%C3%A9es%202.ipynb).