

# NSI - Première - Algorithmie

## Structure imbriquées et compréhension

qkzk

2020/11/22

## Structures imbriquées et compréhension

*Il est possible de combiner listes, tuples et dictionnaires. De plus avec la syntaxe des compréhensions en Python, l'écriture des listes et des dictionnaires est rendue particulièrement compacte et élégante.*

### 1. Les structures imbriquées

#### 1. Construction des structures imbriquées

- On peut **imbriquer** des listes, des tuples et des dictionnaires. La seule règle est qu'une clé de dictionnaire doit être *hachable*, c'est-à-dire récursivement *non mutable*.
- On peut donc construire des listes de listes, des listes de tuples, des listes de dictionnaires, des tuples de listes, des tuples de dictionnaires, des dictionnaires ayant des listes ou des tuples comme *valeurs*.
- On ne peut pas construire des dictionnaires ayant des listes comme *clés* car les listes sont mutables.

#### Exemple :

- Création d'une liste de 3 tuples (représentant des points du plan donnés par leur abscisse et leur ordonnée) :

```
>>> lst = [(4, 5), (-1, 0), (2.5, 1)]
>>> len(lst)
3
>>> lst[1]
(-1, 0)
```

- Chaque élément de la liste est un tuple :

```
>>> t = lst[1]
>>> type(t)
<class 'tuple'>
>>> t[0]
-1
```

- On peut **accéder directement** à l'abscisse ou à l'ordonnée d'un point du plan. Valeur numéro 1 du tuple 2 de la liste :

```
>>> lst[2][1]
1
```

#### 2. Parcours d'une structure imbriquée

- Le parcours d'une structure imbriquée nécessite plusieurs boucles.

#### Exemple :

```
persos = [{"prenom": "Bilbo", "nom": "Baggins", "age": 111},
          {"prenom": "Frodo", "nom": "Baggins", "age": 33},
          {"prenom": "Sam", "nom": "Gamgee", "age": 21}]
```

```
# Parcours de la liste
```

```

for p in persos:
    print("-----")
    # Parcours d'un dict
    for k, v in p.items():
        print(k, '->', v)

```

Qui produit l'affichage ci-dessous :

```

-----
prenom -> Bilbo
nom -> Baggins
age -> 111
-----
prenom -> Frodo
nom -> Baggins
age -> 33
-----
prenom -> Gamgee
nom -> Sam
age -> 21

```

## 2. Les compréhensions

La notation en compréhension permet de créer une liste ou un dictionnaire sans en énumérer explicitement les éléments.

### 1. Listes en compréhension

- Création d'une liste en compréhension

**Exemple** : Liste des nombres entiers de 2 à 10 inclus.

```

>>> [i for i in range(2, 11)] # on peut simplement écrire `list(range(2, 11))`
[2, 3, 4, 5, 6, 7, 8, 9, 10]

```

- Application d'une fonction à chaque élément

**Exemple** : Liste des carrés des nombres entiers de 2 à 10 inclus

```

>>> [i ** 2 for i in range(2, 11)]
[4, 9, 16, 25, 36, 49, 64, 81, 100]

```

- Filtrage des éléments en fonction d'une condition

**Exemple** : Liste des carrés des nombres entiers de 2 à 50 inclus qui se terminent par le chiffre 9.

```

>>> [i ** 2 for i in range(2, 51) if (i ** 2) % 10 == 9]
[9, 49, 169, 289, 529, 729, 1089, 1369, 1849, 2209]

```

### 2. Dictionnaires en compréhension

- Pour les dictionnaires, la syntaxe est équivalente. Il faut préciser la clé et la valeur pour chaque élément.

**Exemple** : Dictionnaire contenant pour les clés, les nombres entiers de 2 à 10 inclus, et pour les valeurs associées, le cube de la clé.

```

>>> {k: k ** 3 for k in range(2, 11)}
{2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343, 8: 512, 9: 729, 10: 1000}

```