# Clés immuables

## Complément - niveau intermédaire

Nous avons vu comment manipuler un dictionnaire, il nous reste à voir un peu plus en détail les contraintes qui sont mises par le langage sur ce qui peut servir de clé dans un dictionnaire.

### Une clé doit être immuable

Si vous vous souvenez de la vidéo sur les tables de hash, la mécanique interne du dictionnaire repose sur le calcul, à partir de chaque clé, d'une fonction de hachage.

C'est à dire que pour simplifier, on localise la présence d'une clé en calculant d'abord

$ f ( clé ) = hash $

puis on poursuit la recherche en utilisant $hash$ comme indice dans le tableau contenant les couples (clé, valeur).

On le rappelle, c'est cette astuce qui permet de réaliser les opérations sur les dictionnaires en temps constant - c'est-à-dire indépendamment du nombre d'éléments.

Cependant, pour que ce mécanisme fonctionne, il est indispensable que la valeur de la clé reste inchangée pendant la durée de vie du dictionnaire. Sinon, bien entendu, on pourrait avoir le scénario suivant:

- on range un tuple `(clef, valeur)` à un premier indice $f(clef) = hash_1 $
- on modifie la valeur de $clef$ qui devient $clef'$
- on recherche notre valeur à l'indice $f(clef') = hash_2 \neq hash_1$

et donc avec ces hypothèses on n'a plus la garantie de bon fonctionnement de la logique.

### Une clé doit être globalement immuable

Nous avons depuis le début du cours longuement insisté sur le caractère mutable ou immuable des différents types prédéfinis de python. Vous devez donc à présent avoir au moins en partie ce tableau en tête:

| Type | |
|---|---|
| `int`,`long`,`float` | immuable |
| `complex`,`bool` | immuable |

| Type | |
|---|---|
| `str` | immuable |
| `list` | mutable |
| `dict` | mutable |
| `set` | mutable |
| `frozenset` | immuable |

Le point important ici, est qu'il **ne suffit pas**, pour une clé, d'être **de type immuable**.

On peut le voir sur cet exemple très simple.

Donnons nous donc un dictionnaire

```
d = {}
```

Et commençons avec un objet de type immuable, un tuple d'entiers

```
bonne_cle = (1, 2)
```

Cet objet est non seulement **de type immuable**, mais tous ses composants et sous-composants sont **immuables**, on peut donc l'utiliser comme clé dans le dictionnaire

```
d[bonne_cle] = "pas de probleme ici"
print d
```

Si à présent on essaie d'utiliser comme clé un tuple qui contient une liste:

```
mauvaise_cle = (1, [1, 2])
```

Il se trouve que cette clé, **bien que de type immuable**, peut être **indirectement modifiée** puisque:

```
mauvaise_cle[1].append(3)
print mauvaise_cle
```

Et c'est pourquoi on ne peut pas utiliser cet objet comme clé dans le dictionnaire

```
d[mauvaise_cle] = 'on ne peut pas faire ceci'
```

Pour conclure, il faut retenir qu'un objet n'est éligible pour être utilisé comme clé que s'il est **composé de types immuables du haut en bas** de la structure de données.

La raison d'être principale du type `tuple`, que nous avons vu la semaine passée, et du type

`frozenset`, que nous verrons très prochainement, est précisément de construire de tels objets globalement immuables.