CC2.0 BY SA – www.infoforall.fr

22 - Le type abstrait DICTIONNAIRE

I – Qu'est-ce que le type abstrait DICTIONNAIRE ?

1.1 Idée générale

Un **dictionnaire** (ou **tableau associatif**) possède les propriétés suivantes :

- il est composé d'un **ensemble de couples (clé, valeur)**.
- **ces couples ne sont pas ordonnés** dans un dictionnaire. Aucun n'est le 1e, le 2e...
- les **clés n'ont pas à être toutes du même type**
- les **valeurs n'ont pas à être toutes du même type**
- on peut accéder à n'importe quelle valeur pourvu qu'on connaisse sa clé
- on peut rajouter ou supprimer n'importe quelle clé et supprimer la valeur associée

1.2 Interface du type abstrait DICTIONNAIRE

Prototype de la fonctions d'interface	Description (cas non mutable)
nouveauDictionnaire() -> Dictionnaire	**dictA =** nouveauDictionnaire() Le contenu de dictA est alors {}
ajouter(cle:Cle, valeur:Valeur, d:Dictionnaire**)** -> None	**ds =** nouveauDictionnaire() **ajouter**("Alice", 12, ds) ds contient {'Alice': 12}
modifier(cle:Cle, valeur:Valeur, d:Dictionnaire) -> None	**modifier**("Alice", 18, ds) ds contient {'Alice': 18}
rechercher(cle:Cle, d:Dictionnaire) -> Valeur	**rechercher**("Alice", ds) *Renvoie 18*
supprimer(cle:Cle, d:Dictionnaire) -> None	**supprimer**("Alice", ds) ds contient {}

II – Implémentation Python

Fonction d'interface	Utilisation en Python (**dict** est un **type mutable**)
nouveauDictionnaire	**ds =** {}
ajouter	**ds["Alice"] = 12**
modifier	**ds["Alice"] = 18**
rechercher	***ds*["Alice"]** *Renvoie 18*
supprimer	Utilisation d'un mot-clé : **del ds["Alice"]** Utilisation d'une méthode : **ds.pop("Alice")**

III – Coûts attendus dans une bonne implémentation

Lecture et modification : coût constant MAIS Recherche : coût linéaire

Insertion et suppression : coût constant Concaténation : coût linéaire

Comme le dictionnaire ne possède pas d'indice permettant de trier les éléments, on ne peut pas utiliser la recherche par dichotomie. On pourrait utiliser des clés valant 0, 1, 2, 3... mais on perd l'intêret d'avoir des clés descriptives et on obtient au final un tableau implémenté de très mauvaise façon.

IV – Parcours d'un dictionnaire Python

Méthode keys() pour obtenir l'ensemble des clés existantes :

```
1     dico = {'Ananas':2, 'Bananes':5, 'Cerises': 7, 'Kiwi':10, 'Poires':4}
2
3     for cle in dico.keys() :
4         print(cle)
```

On obtient : `'Ananas'` puis `'Bananes'` puis `'Cerises'` puis `'Kiwi'` puis `'Poires'`

Comme on récupére les clés, on peut modifier l'état du dictionnaire.

Méthode values() pour obtenir l'ensemble des valeurs existantes :

```
1     dico = {'Ananas':2, 'Bananes':5, 'Cerises': 7, 'Kiwi':10, 'Poires':4}
2
3     for v in dico.values() :
4         print(v)
```

On obtient : `2 puis 5 puis 7 puis 10 puis 4`

Méthode items() pour obtenir l'ensemble des couples (clé, valeur) existants :

```
1     dico = {'Ananas':2, 'Bananes':5, 'Cerises': 7, 'Kiwi':10, 'Poires':4}
2
3     for couple in dico.items() :
4         print(couple)
```

On obtient : `('Ananas', 2) puis ('Bananes', 5) puis ('Cerises', 7)...`

On notera qu'on peut récupérer directement la clé et la valeur avec **couple[0]** et **couple[1]**.

Quelques exemples d'utilisation :

```
def rajoute_2_si_existe(d, v) :
    '''Rajoute 2 à la valeur si elle existe'''
    for (cle, valeur) in d.items() :
        if valeur == v :
            d[cle] = valeur + 2
```

```
def rajoute_2_si_existe(d, v) :
        '''Rajoute 2 à la valeur si elle existe'''
        for cle in d.keys() :
                if d[cle] == v :
                        d[cle] = d[cle] + 2
```

Attention : pour **parcourir**, le dictionnaire les 3 méthodes sont valides.

Par contre, pour **modifier** ; il faut connaître la clé : seules **keys() et items()** le permettent.