## TP : Réalisez un dictionnaire ordonné

Voici enfin le moment de la pratique. Vous avez appris pas mal de choses dans cette partie, beaucoup de concepts, souvent théoriques. Il est temps de les mettre en application, dans un contexte un peu différent des TP précédents : on ne va pas créer un jeu mais plutôt un objet conteneur tenant à la fois du dictionnaire et de la list e.

### Notre mission

Notre énoncé va être un peu différent de ceux dont vous avez l'habitude. Nous n'allons pas créer ici un jeu mais simplement une classe, destinée à produire des objets conteneurs, des dictionnaires ordonnés.

Peut-être ne vous en souvenez-vous pas mais je vous ai dit dans le chapitre consacré aux dictionnaires que c'était un type non-ordonné. Ainsi, l'ordre dans lequel vous entrez les données n'a pas d'importance. On ne peut ni les trier, ni les inverser, tout cela n'aurait aucun sens pour ce type particulier.

Mais nous allons profiter de l'occasion pour créer une forme de dictionnaire ordonné. L'idée, assez simplement, est de stocker nos données dans deux listes :

* la première contenant nos clés ;
* la seconde contenant les valeurs correspondantes.

L'ordre d'ajout sera ainsi important, on pourra trier et inverser ce type de dictionnaire.

#### Spécifications

Voici la liste des mécanismes que notre classe devra mettre en œuvre. Un peu plus bas, vous trouverez un exemple de manipulation de l'objet qui reprend ces spécifications :

1. On doit pouvoir créer le dictionnaire de plusieurs façons :
   * Vide : on appelle le constructeur sans lui passer aucun paramètre et le dictionnaire créé est donc vide.
   * Copié depuis un dictionnaire : on passe en paramètre du constructeur un dictionnaire que l'on copie par la suite dans notre objet. On peut ainsi écrireconstructeur(dictionnaire)et les clés et valeurs contenues dans le dictionnaire sont copiées dans l'objet construit.
   * Pré-rempli grâce à des clés et valeurs passées en paramètre : comme les dictionnaires usuels, on doit ici avoir la possibilité de pré-remplir notre objet avec des couples clés-valeurs passés en paramètre (constructeur(cle1 = valeur1, cle2 = valeur2, …)).
2. Les clés et valeurs doivent être couplées. Autrement dit, si on cherche à supprimer une clé, la valeur correspondante doit également être supprimée. Les clés et valeurs se trouvant dans des listes de même taille, il suffira de prendre l'indice dans une liste pour savoir quel objet lui correspond dans l'autre. Par exemple, la clé d'indice0est couplée avec la valeur d'indice0.
3. On doit pouvoir interagir avec notre objet conteneur grâce aux crochets, pour récupérer une valeur (objet[cle]), pour la modifier (objet[cle] = valeur) ou pour la supprimer (del objet[cle]).
4. Quand on cherche à modifier une valeur, si la clé existe on écrase l'ancienne valeur, si elle n'existe pas on ajoute le couple clé-valeur à la fin du dictionnaire.
5. On doit pouvoir savoir grâce au mot-cléinsi une clé se trouve dans notre dictionnaire (cle in dictionnaire).
6. On doit pouvoir demander la taille du dictionnaire grâce à la fonctionlen.
7. On doit pouvoir afficher notre dictionnaire directement dans l'interpréteur ou grâce à la fonctionprint. L'affichage doit être similaire à celui des dictionnaires usuels ({cle1: valeur1, cle2: valeur2, …}).
8. L'objet doit définir les méthodessortpour le trier etreversepour l'inverser. Le tri de l'objet doit se faire en fonction des clés.
9. L'objet doit pouvoir être parcouru. Quand on écritfor cle in dictionnaire, on doit parcourir la liste des clés contenues dans le dictionnaire.
10. À l'instar des dictionnaires, trois méthodeskeys()(renvoyant la liste des clés),values()(renvoyant la liste des valeurs) etitems()(renvoyant les couples (clé, valeur)) doivent être mises en œuvre. Le type de retour de ces méthodes est laissé à votre initiative : il peut s'agir d'itérateurs ou de générateurs (tant qu'on peut les parcourir).
11. On doit pouvoir ajouter deux dictionnaires ordonnés (dico1 + dico2) ; les clés et valeurs du second dictionnaire sont ajoutées au premier.

Cela vous en fait, du boulot !

Et vous pourrez encore trouver le moyen d'améliorer votre classe par la suite, si vous le désirez.

#### Exemple de manipulation

Ci-dessous se trouve un exemple de manipulation de notre dictionnaire ordonné. Quand vous aurez codé le vôtre, vous pourrez voir s'il réagit de la même façon que le mien.

>>> fruits = DictionnaireOrdonne()

>>> fruits

{}

fruits["pomme"] = 52

fruits["poire"] = 34

fruits["prune"] = 128

fruits["melon"] = 15

>>> fruits

{'pomme': 52, 'poire': 34, 'prune': 128, 'melon': 15}

>>> fruits.sort()

>>> print(fruits)

{'melon': 15, 'poire': 34, 'pomme': 52, 'prune': 128}

>>> legumes = DictionnaireOrdonne(carotte = 26, haricot = 48)

>>> print(legumes)

{'carotte': 26, 'haricot': 48}

>>> len(legumes)

2

>>> legumes.reverse()

>>> fruits = fruits + legumes

>>> fruits

{'melon': 15, 'poire': 34, 'pomme': 52, 'prune': 128, 'haricot': 48, 'carotte':

26}

>>> del fruits['haricot']

>>> 'haricot' in fruits

False

>>> legumes['haricot']

48

>>> for cle in legumes:

... print(cle)

...

haricot

carotte

>>> legumes.keys()

['haricot', 'carotte']

>>> legumes.values()

[48, 26]

>>> for nom, qtt in legumes.items():

... print("{0} ({1})".format(nom, qtt))

...

haricot (48)

carotte (26)

>>>

#### Tous au départ !

Je vous ai donné le nécessaire, c'est maintenant à vous de jouer. Concernant l'implémentation, les fonctionnalités, il reste des zones obscures, c'est volontaire. Tout ce qui n'est pas clairement dit est à votre initiative. Tant que cela fonctionne et que l'exemple de manipulation ci-dessus affiche la même chose chez vous, c'est parfait. Si vous voulez mettre en œuvre d'autres fonctionnalités, méthodes ou attributs, ne vous gênez pas… mais n'oubliez pas d'y aller progressivement.

C'est parti !