T.P. Dictionnaires

NSI Terminale

Rappels sur les dictionnaires

Dans cette partie, nous allons fabriquer un carnet d'adresse pour stocker des contacts.

Fabrication d'un contact

Chaque contact sera un dictionnaire dont les clés seront : - `nom` : Nom et prénom du contact - `tel` : N° de téléphone - `rue` : adresse complète - `code` : code postal - `ville` : ville

Créez un dictionnaire nommé `contact` correspondant au contact suivant : > Margaret Costa-Royer > 08 06 18 37 28 > 93, avenue Bruneau > 13749 Perrot

```
# YOUR CODE HERE
```

```
# Vérification
assert contact["nom"] == "Margaret Costa-Royer"
assert contact["tel"] == "08 06 18 37 28"
assert contact["ville"] == "Perrot"
```

Ajouter une nouvelle entrée "passwd" dans le contact ayant pour valeur 's75JWikE&o'

```
# YOUR CODE HERE
```

```
# Vérification
assert contact["passwd"] == 's75JWikE&o'
```

Génération automatique d'un contact

Écrire une fonction `genere_contact()` - qui ne prend aucun paramètre - qui renvoie un dictionnaire possédant les mêmes clés que le contact ci-dessus, y compris "passwd"

On pourra utiliser le module `faker` de python dont un exemple d'utilisation est donné dans la cellule ci-dessous.

```
%pip install faker
```

```
from faker import Faker

fake = Faker("fr_FR") # Générateur de données personnelles pour un français

print(fake.name())
print(fake.phone_number())
print(fake.street_address())
print(fake.postcode(), fake.city())
print(fake.password())
```

Remarque : Si la cellule ci-dessus provoque une erreur disant que le module `faker` n'est pas disponible, vous pouvez l'installer dans l'environnement jupyter via la commande

`!pip install faker`

```
def genere_contact():
    """Fabrique un contact factice et renvoie le contact sous forme d'un dictionnaire"""
    # YOUR CODE HERE
```

```
contact1 = genere_contact()
assert type(contact1["nom"]) == str
assert "ville" in contact1
```

Mise en pratique

Fabrication du carnet d'adresse

Tout est à présent en place pour que nous puissions fabriquer notre carnet d'adresse.

Première implémentation

Dans une première approche, nous allons considérer que le carnet d'adresse sera une liste de contacts, chaque contact étant un dictionnaire dont la structure a été définie à la section précédente.

Fabriquez une fonction `genere_carnet1` - prenant en paramètre le nombre `n` de contacts à générer - renvoyant une **liste** de `n` contacts générés aléatoirement.

```
def genere_carnet1(n):
    """Renvoie une liste de n contacts aléatoires"""
    # YOUR CODE HERE
```

```
# vérification

carnet1 = genere_carnet1(10)
assert type(carnet1) == list
assert "nom" in carnet1[3]
```

Ecrire à présent une fonction `est_present` - prenant 2 paramètres : un nom et un carnet d'adresse - renvoyant `True` si le nom figure dans le carnet d'adresse, `False` sinon

```
def est_present(nom, carnet):
    """Teste si nom est présent dans le carnet d'adresse"""
    # YOUR CODE HERE
```

```
# Vérification

carnet1 = genere_carnet1(10)
nom = carnet1[-1]["nom"]
assert est_present(nom, carnet1)
assert not est_present("Géo Trouvetout", carnet1)
```

Mesure de performance de la recherche

Nous allons regarder ici comment évolue la vitesse de recherche en fonction de la taille du carnet d'adresse. On utilisera pour cela la fonction magique de *jupyter* : `%%timeit`. Étudiez la cellule suivante :

```
# Fabrication d'un carnet de 100 contacts
carnet1 = genere_carnet1(100)
nom = carnet1[-1]["nom"]  # On récupère un nom du carnet
nom
```

```
%%timeit

# On mesure le temps d'une recherche
est_present(nom, carnet1)
```

Vous lisez sous la cellule le temps de recherche.

A présent, on refait l'expérience pour 1000 contacts dans le carnet d'adresse.

```
carnet1 = genere_carnet1(1000)
nom = carnet1[-1]["nom"]  # On récupère un nom du carnet
nom
```

```
%%timeit

# On mesure le temps d'une recherche dans ce carnet
est_present(nom, carnet1)
```

Seconde implémentation

Vous devez avoir constaté ci-dessus que le temps de recherche est proportionnel à la taille du carnet d'adresse : si celui-ci contient 10 fois plus de contact, la recherche peut être jusqu'à 10 fois plus longue.

Nous allons changer d'approche et fabriquer un carnet d'adresse sous forme d'un dictionnaire dont les clés seront les **noms** et les valeurs seront les fiches contacts. Ainsi notre carnet d'adresse sera un dictionnaire dont les valeurs seront des dictionnaires !

Fabriquez une fonction `genere_carnet2` - prenant en paramètre le nombre n de contacts à générer - renvoyant un **dictionnaire** de `n` contacts générés aléatoirement

```
def genere_carnet2(n):
    """Renvoie un dictionnaire de n contacts aléatoires"""
    # YOUR CODE HERE
```

```
# Vérification

carnet2 = genere_carnet2(10)
assert type(carnet2) == dict
nom = list(carnet2.keys())[-1]
assert type(carnet2[nom]) == dict
```

Mesure de performance de la recherche

Nous allons regarder pour cette nouvelle implémentation comment évolue la vitesse de recherche en fonction de la taille du carnet d'adresse. Validez les 2 cellules suivantes.

```
# Fabrication d'un carnet de 100 contacts
carnet2 = genere_carnet2(100)
nom = list(carnet2.keys())[-1] # On récupère un nom du carnet
nom
```

```
%%timeit

nom in carnet2                  # On le recherche
```

On constate déjà que la recherche est plus rapide que pour la première implémentation du carnet à l'aide d'un tableau.

Refaisons l'expérience avec 100 fois plus de contacts dans le carnet !!

```
# Fabrication d'un carnet de 10000 contacts
carnet2 = genere_carnet2(10000)
nom = list(carnet2.keys())[-1] # On récupère un nom du carnet
nom
```

```
%%timeit

nom in carnet2            # On le recherche
```

Conclusion

Vous le constatez d'après les expériences ci-dessus : le temps de recherche dans le dictionnaire est pratiquement indépendant du nombre d'entrées dans ce dictionnaire, car en multipliant le nombre de contacts par 100, le temps est resté pratiquement identique alors que dans le cas de la recherche dans un tableau, celui-ci est proportionnel à la longueur du tableau.

Le dictionnaire est donc une structure de données optimisée pour la recherche sur les clés.