Dictionnaires

1. Structure et construction

Structure d'un dictionnaire

Un dictionnaire est un ensemble **non-ordonné** d'associations clé/valeur. On ne peut **pas** accéder à un élément par un indice contrairement aux listes.

Chaque élément de dictionnaire est composé d'une **clé** et d'une **valeur**. Les éléments d'un même dictionnaire sont séparés par des **virgules** et encadrés par des **accolades**.
Les clés d'un dictionnaire sont des objets uniques et ne peuvent être modifiées (**non-mutables**). Par conséquent, elles ne peuvent être que de types **_int_**,**_float_**, **_str_** ou **_tuple_**.
Il n'y a pas de restriction de type pour les valeurs.

exemples de dictionnaires avec des valeurs simples ou multiples :

```
In [ ]: placard={'chemise':3,'pantalon':6,'tee-shirt':7}
        print('placard est de type :',type(placard),' et contient :',placard)
        dressing={'chemise':['1','blanc','étagère du haut'],'pantalon':['8','vert','étagère du milieu']}
        print('dressing est de type :',type(dressing),' et contient :',dressing)
```

Création d'un dictionnaire

Créer un dictionnaire par instanciation avec les couples clé-valeur

exemple :

```
In [ ]: dimpson={"père":"Homer","mère":"Marge","fille":"Bart","fille":"Lisa"}
        print('dimpson contient :',dimpson)
```

application :
Créer un dictionnaire appelé annee dans lequel les clés correspondent aux noms des mois et leurs valeurs associées aux nombres du jour dans le mois (pour une année non-bissextile).

```
In [ ]:
```

Créer un dictionnaire par compréhension

exemple 1 : dictionnaire des carrés des entiers de 0 à 10

```
In [ ]: carre={x:x**2 for x in range(11)}
        print('carre contient : ',carre)
```

exemple 2 : avec les lettres de l'alphabet

```
In [ ]: majuscules={chr(65+i) : i for i in range(26)}
        print(majuscules)
```

Convertir une liste de listes à 2 éléments en dictionnaire

```
In [ ]: stylos=[['bleu',3],['noir',4],['rouge',2],['vert',1]]
        d_stylos=dict(stylos)
        print(d_stylos)
```

Créer un dictionnaire vide

```
In [ ]: dico_vide={}
        print('dico_vide contient : ',dico_vide)
```

2. Opérations sur les dictionnaires

Ajouter des éléments à un dictionnaire

par clé-valeur : nom_du_dico["clé"]=valeur

```
In [ ]: dico_vide["une clé"]="sa valeur"
        print('dico_vide : ',dico_vide)
        placard["tiroir"]=9
        print('placard : ', placard)
        dimpson["prénom"]=["Jason","Jasone","tiroir"]
        print('dressing :',dressing)
        dimpson["chat"]="Garfield"
        print('dimpson :',dimpson)
```

ajouter des valeurs à une clé existante : nom_du_dico["clé"].append(valeur_supplémentaire)

```
In [ ]: dressing["chaussettes"].append("commode")
        print('dressing :',dressing)
```

Remarque : cette fonction ne s'applique que sur des clés dont les valeurs supportent la fonction _append_ , sinon cela renvoie un message d'erreur.

```
In [ ]: dimpson["chat"].append("roux")
```

Récupérer une valeur associée à une clé

```
In [ ]: print('Le chat des Dimpson est ",dimpson["chat"])
```

Remarque : cette méthode renvoie une erreur si la clé n'existe pas dans le dictionnaire.

```
In [ ]: print("Le chien des Dimpson est ",dimpson["chien"])
```

Conséquence : pour éviter le déclenchement de l'erreur, on peut utiliser la fonction nom_du_dico.get("clé") qui renverra None si la clé n'existe pas et la valeur sinon.

```
In [ ]: print("Le chat des Dimpson est ",dimpson.get("chat"))
        print("Le chien des Dimpson est ",dimpson.get("chien"))
```

Modifier des éléments d'un dictionnaire

modifier une valeur par affectation : nom_du_dico["clé"]=nouvelle_valeur

```
In [ ]: print('avant modification, dico_vide : ',dico_vide)
        dico_vide["une clé"]="nouvelle valeur"
        print('après modification, dico_vide : ',dico_vide)
```

Supprimer des éléments d'un dictionnaire

par clé-valeur : del nom_du_dico["clé"]

```
In [ ]: del dico_vide["une clé"]
        print('dico_vide : ',dico_vide)
```

Remarque : cette méthode déclenche une erreur si la clé n'existe pas. On peut tut préférer la méthode suivante qui renvoie la valeur supprimée :

avec la fonction pop() : nom_du_dico.pop("clé")

```
In [ ]: nom_chat=dimpson.pop("chat")
        print('dimpson : ',dimpson)
        print('valeur enlevée : ',nom_chat)
```

vider un dictionnaire : nom_du_dico.clear()

```
In [ ]: placard.clear()
        print('placard : ', placard)
```

Fonctionnalités sur les éléments d'un dictionnaire

nombre de clés : len(nom_du_dico)

```
In [ ]: print('dressing a ',len(dressing),' clés.')
```

accès aux clés d'un dictionnaire : nom_du_dico.keys()

```
In [ ]: print('les clés de dressing sont : ',dressing.keys())
        print(type(dressing.keys()))
```

Remarque : les clés ne sont pas renvoyées sous forme de liste mais sous forme d'un objet particulier, dict_keys, contenant leur liste en argument.
Le type dictionnaire est optimisé pour rechercher efficacement une clé particulière ; le test se fera simplement comme suit :

tester si une clé est dans le dictionnaire :

```
In [ ]: if "chaussettes" in dressing :
            print('les chaussettes sont dans le dressing')
        if not "chaussettes" in placard :
            print("les chaussettes ne sont pas dans le placard")
```

accès aux valeurs du dictionnaire : nom_du_dico.values()

```
In [ ]: print('les valeurs de dimpson sont : ',dimpson.values())
        print(type(dimpson.values()))
```

tester si une valeur est dans le dictionnaire :

```
In [ ]: if not "Garfield" in dimpson.values() :
            print('Garfield n'est plus chez les dimpson,')
        if "Bart" in dimpson.values() :
            print('mais Bart y est encore.')
```

tester si un couple clé-valeur est dans le dictionnaire :

```
In [ ]: if ("rouge",2) in d_stylos.items() :
            print('j'ai 2 stylos rouges, ')
        if not ("vert",4) in d_stylos.items() :
            print('mais je n'ai pas 4 stylos verts.')
```

boucler sur les clés et valeurs : nom_du_dico.items()

```
In [ ]: for key,value in dimpson.items() :
            print(key," vaut ",value)
```

Remarque : les couples clés-valeurs sont de type tuple, type que l'on verra ultérieurement.

3. Applications

Exercice 1 : jouons au scrabble

Au scrabble, chaque lettre a une valeur comprise entre 1 et 10. Compléter le code suivant qui demande à l'utilisateur le mot qu'il veut écrire (tout en majuscules et sans accent) et affiche le score correspondant (en supposant qu'il ne le pose pas sur une case apportant des points supplémentaires).
Modifier le code pour que qu'il prenne en compte le nombre de lettres de chaque type à disposition et affiche impossible au lieu du score s'il n'y a pas assez de lettres disponibles (on pourra intégrer au besoin une lettre joker = lettre blanche).

```
In [ ]: scrabble={'A':[1,9],'B':[3,2],'C':[3,2],'D':[2,3],'E':[1,15],
                  'F':[4,2],'G':[2,2],'H':[4,2],'I':[1,8],'J':[8,1],
                  'K':[10,1],'L':[1,5],'M':[2,3],'N':[1,6],'O':[1,6],
                  'P':[3,2],'Q':[8,1],'R':[1,6],'S':[1,6],
                  'T':[1,6],'U':[4,6],'V':[4,2],'W':[10,1],'X':[10,1],
                  'Y':[10,1],  '0':[0,2]}
```

Exercice 2 : dépouillement d'une urne

A l'issue d'une élection à scrutin uninominal (un seul nom par bulletin), on récupère un tableau contenant tous les noms inscrits sur les bulletins trouvés dans l'urne.
Afin de déterminer le vainqueur de l'élection, en un seul parcours des bulletins, on souhaite créer un dictionnaire qui à chaque nom contenu dans l'urne associe le nombre de bulletins correspondant et se servir de ce dictionnaire pour déterminer le vainqueur. Pour ce faire, on va :

- créer une fonction "incr_dico()" qui ajoute à un dictionnaire passé en argument une clé telle aussi passée en argumentassociée à la valeur 1 si cette clé n'est pas déjà présente dans le dictionnaire , sinon, incrémente de 1 la valeur associée à cette clé.
- créer une fonction "depouillement()" qui prend la liste des bulletins en argument, construit le dictionnaire et retourne le vainqueur.

```
In [ ]: bulletins=['Martin','Paul','Pierre','Jacquez','Toto','Marie','Paul','Pierre','Jacquez','Toto','Toto']

        def incr_dico(dico,cle):
            """ Entrée : un dictionnaire dico
                         une clé cle

                Sortie : rien (c'est une procédure qui modifie le dico existant)
                Effets :
                         si cle est une clé de dico, la valeur de cle est augmentée de 1
                         - sinon cle est insérée dans dico associée à la valeur 1
            """
            #
            # à compléter


        def depouillement(bulletins):
            """ Entrée : une liste bulletins
                Sortie : une chaîne de caractères : vainqueur
                Effets :
                    - construit le dictionnaire des noms-nb de bulletins
                    - détermine le vainqueur
            """
            dico = {}
            ## création du dictionnaire
            #
            # à compléter
            #


            nb_voix_max = 0
            vainqueur=""
            ## détermination du vainqueur
            #
            # à compléter
            #

            return vainqueur

        print('le vainqueur est ',depouillement(bulletins))
```

Exercice 3 : horaires de trains

Le panneau de la gare affiche les horaires suivants :

20:29	**Brest** TGV INOUI N°8641 À l'heure	20:51	**Messac - Guipry** Train TER N°855081 À l'heure
20:32	**Quimper** TGV INOUI N°8720 À l'heure	21:29	**Brest** TGV INOUI N°8631 À l'heure
20:43	**Saint-Malo** Train TER N°854439 À l'heure	21:35	**Paris Montparnasse** TGV INOUI N°8845 Retard estimé à 10 min
20:43	**Saint-Brieuc** Train TER N°854441 À l'heure	21:35	**Paris Montparnasse** TGV INOUI N°8756 Retard estimé à 10 min
20:46	**Montreuil-sur-Ille** Train TER N°854443 À l'heure	21:39	**Saint-Malo** Train TER N°854445 À l'heure

On peut les regrouper dans un dictionnaire de dictionnaires.
Les 3 premiers affichages ont ainsi été répertoriés dans le dictionnaire trains ci-dessous.
Travail à effectuer :

- compléter la fonction ajout_train() qui ajoute au dictionnaire un train indexé par son numéro et dont la valeur est un dictionnaire comportant les champs destination, horaire, type et délai dont les valeurs ont été passées en argument de la fonction ;
- compléter la fonction prochain_train() qui prend en argument le dictionnaire des trains et une destination et retourne l'horaire du prochain train concerné si cette destination existe et None sinon.
- compléter la fonction ajout_train_interactif() qui demande à l'utilisateur les informations sur le train à rajouter (numéro, destination, horaire, type et délai) et l'ajoute au dictionnaire ;
- créer une fonction train_pourQ qui prend en argument le dictionnaire des trains, demande à l'utilisateur la destination souhaitée et lui affiche l'horaire si cette destination existe et un message sinon.

```
In [ ]: trains={8641:{'destination':'Brest',
                       'horaire':'20:29',
                       'type':'TGV INOUI',
                       'délai':'à l\'heure'},
                8739:{'destination':'Quimper',
                       'horaire':'20:32',
                       'type':'TGV INOUI',
                       'délai':'à l\'heure'},
                8544:{'destination':'Saint-Malo',
                       'horaire':'20:43',
                       'type':'TER',
                       'délai':'à l\'heure'}}

        def ajout_train(train,numero,destination,horaire,genre,ponctualite):
            """ Entrée : le dictionnaire des trains
                         des arguments précisant pour le train
                         numéro, destination, horaire, type de train, délai
                Sortie : rien (le dictionnaire est modifié in situ)
                Effets : rajout du train au dictionnaire des trains
            """
            #
            # à compléter
            #

        def prochain_train(trains,destination):
            """ Entrée : le dictionnaire des trains
                         la destination souhaitée

                Sortie :
                    - l'horaire le plus tôt si la destination existe
                    - None sinon
                Effets : rajout du train au dictionnaire des trains
            """
            #
            # à compléter
            #


        def ajout_train_interactif(trains):
            """ Entrée : le dictionnaire des trains
                         Sortie : rien (le dictionnaire est modifié in situ)
                Effets :
                        - demande à l'utilisateur de rentrer les références du train :
                        numéro, destination, horaire, type de train, délai
                    - rajout du train correspondant au dictionnaire des trains
            """
            #
            # à compléter
            #


        ### programme principal
        ######################
        ## partie à ne pas modifier

        ajout_train(trains,855607,'Saint-Brieuc',20:43,'TER','à l\'heure')
        ajout_train(trains,854289,'Montreuil-sur-Ille',20:46,'TER','à l\'heure')
        ajout_train(trains,856089,'Messac-Guipry',20:51,'TER','à l\'heure')
        ajout_train(trains,8651,'Brest',21:29,'TGV INOUI','à l\'heure')

        ajout_train_interactif(trains)
        ajout_train_interactif(trains)
        ajout_train_interactif(trains)


        ## partie éventuellement modifiable

        print('le prochain train à destination de Brest est à ',prochain_train(trains,'Brest'))
```

```
In [ ]:
```