08-cours_python_biost1_cent1_dict

April 2, 2020

1 Les dictionnaires

1.1 Présentation

- Un dictionnaire est une structure de données permettant de mémoriser des couples : clé - valeur. Il est très utilisé en langage Python

Exemple :

```
fiche_contact = {
    'prenom': 'Lucie',
    'nom': 'Fer',
    'tel': '0102030405',
}
```

- Les valeurs associées aux clés peuvent être de différent type :

```
fiche_contact = {
    'prenom': 'Lucie',
    'nom': 'Fer',
    'tel': '0102030405',
    'age' : 90,
    'hobbies': ['Surf', 'Rando', 'Python']
}
```

- On accède à la valeur en spécifiant la clé entre **crochets** :

```
fiche_contact = {
    'prenom': 'Lucie',
    'nom': 'Fer',
    'tel': '0102030405',
    'age' : 90,
    'hobbies': ['Surf', 'Rando', 'Python']
}
prenom_contact = fiche_contact['prenom']
print(prenom_contact, type(prenom_contact))
hobbies_contact = fiche_contact['hobbies']
print(hobbies_contact, type(hobbies_contact))
```

Résultat :

```
Lucie <class 'str'>
['Surf', 'Rando', 'Python'] <class 'list'>
```

1.1.1 Exercice : "Nomenclature and Symbolism for Amino Acids and Peptides"

```
Ala, Alanine
Arg, Arginine
Asn, Asparagine
...
Val, Valine
Asx, Aspartic acid or Asparagine
Glx, Glutamine or Glutamic acid
```

- Créer un dictionnaire pour mémoriser ces données
- Afficher le dictionnaire
- Afficher le descriptif correspondant à 'Asx'

```
In [8]: amino_acids_peptides = {
            'Ala': 'Alanine',
            'Arg': 'Arginine',
            'Asn': 'Asparagine',
            'Val': 'Valine',
            'Asx': 'Aspartic acid or Asparagine',
            'Glx': 'Glutamine or Glutamic acid',
        }

        print(amino_acids_peptides)
        print('Asx', ":", amino_acids_peptides['Asx'])
        # Remarque : autre possibilité pour **l'affichage**
        print(f"Asx : {amino_acids_peptides['Asx']}")
```

```
{'Ala': 'Alanine', 'Arg': 'Arginine', 'Asn': 'Asparagine', 'Val': 'Valine', 'Asx': 'Aspartic ac
Asx : Aspartic acid or Asparagine
Asx : Aspartic acid or Asparagine
```

1.1.2 Même exercice mais en demandant à l'utilisateur de saisir une clé et lui afficher la description correspondante

- On traitera le cas où la clé saisie n'existe pas

```
In [3]: amino_acids_peptides = {
            'Ala': 'Alanine',
            'Arg': 'Arginine',
            'Asn': 'Asparagine',
            'Val': 'Valine',
            'Asx': 'Aspartic acid or Asparagine',
```

```
            'Glx': 'Glutamine or Glutamic acid',
        }
        cle = input("Clé : ")
        if cle not in amino_acids_peptides:
            print(f"{cle} : clé inconnue")
        else:
            print(amino_acids_peptides[cle])
```

```
Clé :Val
Valine
```

1.1.3 Exercice : PIB / habitant en euro des pays en Europe (2017) :

```
Luxembourg  107865.27
Norvège     91218.62
Suisse  76667.44
Irlande     74433.46
Danemark    61582.17
```

- Créer un dictionnaire pour mémoriser ces données
- Afficher le dictionnaire
- Afficher le PIB / hab. de la Suisse

```
In [6]: pib_hab = {
            'Luxembourg': 107865.27,
            'Norvège': 91218.62,
            'Suisse': 76667.44,
            'Irlande': 74433.46,
            'Danemark': 61582.17,
        }

        print(pib_hab)
        pib_suisse = pib_hab['Suisse']
        print(f"Suisse : {pib_suisse} ")
```

```
{'Luxembourg': 107865.27, 'Norvège': 91218.62, 'Suisse': 76667.44, 'Irlande': 74433.46, 'Danema
Suisse : 76667.44
```

1.1.4 Même exercice mais en demandant à l'utilisateur de saisir un pays et lui afficher le PIB / hab. correspondant

- On traitera le cas où le pays saisi n'existe pas (dans le dictionnaire)

```
In [1]: pib_hab = {
            'Luxembourg': 107865.27,
            'Norvège': 91218.62,
            'Suisse': 76667.44,
```

```
            'Irlande': 74433.46,
            'Danemark': 61582.17,
        }
        pays = input("Nom du pays : ")
        pays = pays.capitalize()
        if pays not in pib_hab:
            print("Ce pays n'existe pas dans les données")
        else:
            print(f"{pib_hab[pays]} ")
```

```
Nom du pays : irlande
74433.46
```

1.2 Affectation dynamique des valeurs d'un dictionnaire

```
fiche_contact = {}  # initialisation
fiche_contact['prenom'] = 'Lucky'  # 'prenom' est la clé, 'Lucky' la valeur
fiche_contact['nom'] = 'Luke'
print(fiche_contact)
```

- Résultat de l'instruction `print`: `{'prenom': 'Lucky', 'nom': 'Luke'}`
- Notez l'utilisation des accolades et des crochets en fonction de l'opération qu'on souhaite effectuer

1.2.1 Exercice : stockage d'éléments dans un dictionnaire

- Demander à l'utilisateur de rentrer un prénom, un nom et une ville
- Stocker ces valeurs dans un dictionnaire
- Afficher le dictionnaire

```
In [3]: donnees_u = {}
        prenom = input("Prénom :")
        # Création d'un élément dans donnees_u
        donnees_u['prenom'] = prenom
        nom = input("Nom :")
        # Création d'un autre élément dans donnees_u
        donnees_u['nom'] = nom
        ville = input("Ville :")
        donnees_u['ville'] = ville
        print(donnees_u)
```

```
Prénom :Marc
Nom :Schneider
Ville :Brest
{'prenom': 'Marc', 'nom': 'Schneider', 'ville': 'Brest'}
```

1.2.2 Exercice : manipulation de plusieurs dictionnaires

En 1982 la population des villes de Brest et Quimper était la suivante :

```
d_hab_1982 = {
    'brest': 156060,
    'quimper': 56907,
}
```

En 2016, Brest a désormais 6426 habitants de moins et Quimper 6498 de plus. - Créer un nouveau dictionnaire `d_hab_2016` dans lequel on mettra à jour le nombre d'habitants, en utilisant les valeurs de `d_hab_1982` - Afficher les deux dictionnaires

```
In [ ]: # Remarque : attention lors de la copie des dictionnaires. Le code ci-dessous fonctions

        d_hab_1982 = {
            'brest': 156060,
            'quimper': 56907,
        }
        d_hab_2016 = d_hab_1982
        # En mettant à jour d_hab_1996 on met aussi à jour d_hab_1982
        # Affecter un dictionnaire dans un autre ne crée donc pas de copie !
        d_hab_2016['brest'] = d_hab_2016['brest'] - 6426
        d_hab_2016['quimper'] = d_hab_2016['quimper'] + 6498
        # Affichage du résultat
        print("Données 1982 : ", d_hab_1982)
        print("Données 2016 : ", d_hab_2016)


        # Résultat :

        # Données 1982 :  {'brest': 149634, 'quimper': 63405}
        # Données 2016 :  {'brest': 149634, 'quimper': 63405}
```

1.2.3 Types de valeurs dans les dictionnaires

- Dans un dictionnaire, on peut mettre dans les valeurs, n'importe quel objet Python : chaine, entier, float… mais aussi des éléments plus complexes : listes, tuples, sets. Ainsi si je souhaite mémoriser les notes des différentes promos dans un dictionnaire, je peux écrire par exemple :

```
notes = {
    'biost1': [18, 19, 14, 15],
    'cent_est1' : [15, 15, 17, 18],
}

notes_biost1 = notes['biost1']
print(notes_biost1)
print(type(notes_biost1))
```

Résultat :

```
[18, 19, 14, 15]
<class 'list'>
```

- Inversement, je peux stocker les dictionnaires dans des suites d'éléments comme les listes, par exemple :

```
fiche_contact1 = {
    'prenom': 'Lucky',
    'nom': 'Luke',
    'ville': 'Chicago'
}
fiche_contact2 = {
    'prenom': 'Joe',
    'nom': 'Dalton',
    'ville': 'Jail City',
}

contacts = [fiche_contact1, fiche_contact2]
for contact in contacts:
    print(contact['nom'])
```

Résultat :

```
Luke
Dalton
```

1.2.4 Types de clés dans les dictionnaires

Les types possibles de clés sont les types "de base" de Python : str, int, float.

```
d1 = {
    'prenom' : "John",
    'nom' : "Wayne",
}

d2 = {
    0 : "John",
    1 : "Wayne",
}
# La clé est de type 'str'
print(d1['prenom'])
# La clé est de type 'int'
# Rien à avoir ici avec l'index 0 d'une liste
print(d2[0])
```

Résultat :

```
John
John
```

1.3 Exercice : enregistrement de dictionnaires dans une liste

- Demander à l'utilisateur de rentrer un prénom, un nom et une ville
- Stocker ces valeurs dans un dictionnaire
- Ajouter ce dictionnaire à une liste
- Répéter l'opération tant que l'utilisateur ne saisit pas une valeur vide pour le prénom
- Afficher le contenu de la liste à la fin de la saisie

```
In [10]: liste_personnes = []
         while True:
             prenom = input("Prénom :")
             if prenom == "":
                 break
             info_personne = {}
             info_personne['prenom'] = prenom
             info_personne['nom'] = input("Nom :")
             info_personne['ville'] = input("Ville :")
             liste_personnes.append(info_personne)
         print(liste_personnes)

         # Que se passe t-il si on initialise *info_personne* **avant** la boucle *while* ?
```

```
Prénom :Alain
Nom :Terrieur
Ville :Quimper
Prénom :Alex
Nom :Terrieur
Ville :Brest
Prénom :
[{'prenom': 'Alain', 'nom': 'Terrieur', 'ville': 'Quimper'}, {'prenom': 'Alex', 'nom': 'Terrieu
```

1.4 Exercice : structurer des données

Soient les listes :

- `lst_etu = ['Thierry', 'Marc', 'Erwan']`
- `lst_notes = [19, 18, 7]`

Créer un dictionnaire pour stocker ces deux listes de façon à ce que la clé soit le prénom de l'étudiant et la valeur sa note.

```
In [2]: lst_etu = ['Thierry', 'Marc', 'Erwan']
        lst_notes = [19, 18, 7]
        d_etu = {}
        for i, etudiant in enumerate(lst_etu):
            d_etu[etudiant] = lst_notes[i]
        print(d_etu)
```

```
{'Thierry': 19, 'Marc': 18, 'Erwan': 7}
```

1.5 Parcourir un dictionnaire

On peut parcourir : - Les clés d'un dictionnaire - Les valeurs d'un dictionnaire - Les deux en même temps

```
ages_personnes = {
    'Enora': 12,
    'Erwan': 83,
    'Gael' : 44,
}

print("# Affichage clés")
for prenom in ages_personnes.keys():
    print(prenom)
print("# Affichage valeurs")
for age in ages_personnes.values():
    print(age)
print("# Les deux mon capitaine")
for prenom, age in ages_personnes.items():
    print(prenom, age)
```

Résultat :

```
# Affichage clés
Enora
Erwan
Gael
# Affichage valeurs
12
83
44
# Les deux mon capitaine
Enora 12
Erwan 83
Gael 44
```

1.5.1 Exercice

Soit le dictionnaire suivant :

```
ages_personnes = {
    'Enora': 12,
    'Erwan': 83,
    'Gael' : 44,
}
```

- Récupérer tous les âges, les mettre dans une liste et la trier (par ordre croissant)

```
In [6]: ages_personnes = {
            'Enora': 12,
```

```
            'Erwan': 83,
            'Gael' : 44,
        }
        liste_ages = []
        for age in ages_personnes.values():
            liste_ages.append(age)

        liste_ages = sorted(liste_ages)
        print(liste_ages)
```

```
[12, 44, 83]
```

1.6 Remarque : dictionnaire en argument de fonction

Comme pour les listes, il ne faut faire attention si on modifie le contenu d'un dictionnaire qui est passé en paramètre, sous peine "d'effets de bord" (après l'appel de la fonction dans le programme principal de dictionnaire n'aura plus sa valeur originale)

```
# Rappel avec un type de base le paramètre n'est pas modifié par la fonction
def ma_fonction(mon_int):
    mon_int = 2

a = 3
ma_fonction(a)
print("Après l'appel de la fontion", a)
```

Résultat :

```
Après l'appel de la fontion 3
```

En revanche avec un `dict` :

```
def ma_fonction(mon_dict):
    mon_dict['prenom'] = 'Guy'

d = {'prenom' : "Emile"}
ma_fonction(d)
print("Après l'appel de la fonction", d)
```

Résultat :

```
Après l'appel de la fonction {'prenom': 'Guy'}
```

1.7 Mise à jour d'un dictionnaire

Pour mettre à jour un dictionnaire existant avec un autre dictionnaire, on peut utiliser la méthode `update(...)`.

Exemple :

```
d_etu = {
    'Thierry' : 19,
    'Marc' : 18,
    'Erwan': 7,
}
d_etu_f = {
    'Marie' : 20,
    'Gisèle' : 2,
}
d_etu.update(d_etu_f)
print(d_etu)
```

Résultat :

```
{'Thierry': 19, 'Marc': 18, 'Erwan': 7, 'Marie': 20, 'Gisèle': 2}
```

1.8 Exercice : stockage des notes des étudiants

Dans une célèbre école, on souhaite mémoriser les notes des étudiants sachant que :

- Chaque étudiant est caractérisé par un no. d'étudiant, un prénom et un nom
- Il y a deux semestres (s1 et s2)
- Il peut y avoir plusieurs notes par semestre

Proposer une structure permettant de stocker ces informations :

- Faire un exemple avec trois étudiants
- Pour les tests, dans le programme principal, demander à l'utilisateur le no. d'étudiant et le semestre et afficher ensuite les notes

Indications :

- Créer un dictionnaire *etudiants* qui contiendra tous les étudiants
- Pour chaque étudiant :
  - Créer un dictionnaire *info* pour les informations de chaque étudiant (nom, prénom)
  - Créer un dictionnaire *notes* pour les notes (s1 et s2)

```
In [1]: etudiants = {
            '1' : {
                'info' : {
                    'prenom' : 'Lucky',
                    'nom' : 'Luke',
                },
                'notes' : {
                    's1' : [11, 13],
                    's2' : [15, 19],
                }
            },
            '2' : {
```

```
            'info' : {
                'prenom' : 'Joe',
                'nom' : 'Dalton',
            },
            'notes' : {
                's1' : [10, 7],
                's2' : [6, 13],
            }
        },
        '3' : {
            'info' : {
                'prenom' : 'Jolly',
                'nom' : 'Jumper',
            },
            'notes' : {
                's1' : [14, 15],
                's2' : [16, 14],
            }
        },
    }

    # PP
    id_etudiant = input("Id de l'étudiant :")
    # etudiant et etudiants sont tous les deux de type dict
    # edudiant est un "sous-dictionnaire" de etudiants
    etudiant = etudiants[id_etudiant]
    if etudiant is None:
        print("Etudiant inexistant")
    else:
        semestre = input("Semestre :")
        notes_semestre = etudiant['notes'][semestre]
        if notes_semestre is None:
            print("Semestre inexistant")
        else:
            print(f"{etudiant['info']['prenom']} {etudiant['info']['nom']} : {notes_semesti
```

```
Id de l'étudiant :2
Semestre :s2
Joe Dalton : [6, 13]
```

1.9 Exercice : modélisation d'un échiquier

1.9.1 Présentation

Dans cet exercice on cherche un moyen efficace et élégant de modéliser un échiquier. Dans un jeu d'échec un plateau est composé de 64 cases. Horizontalement sont définies les lettres de `a` à `h` et verticalement les chiffres de `1` à `8`.

On se propose ici d'utiliser un dictionnaire qui permettra de mémoriser quelle pièce (roi, reine, etc.) est présente à quelle case.

On peut utiliser des tuples de deux éléments comme clé de dictionnaire. Les tuples sont des séquences d'éléments (comme les listes) mais ils ne sont pas modifiables et donc ils peuvent êtres utilisés comme clé.

Exemple :

```
valeurs = (1, 2)
valeurs[0]  # donne 1
valeurs[0] = 3  # erreur, on ne peut pas modifier
```

Pour notre exercice on pourrait avoir un dictionnaire qui aurait la forme suivante :

```
echiquier = {
    ('a', 1): 'Roi',
    ('a', 2): None,
    ...
    ('d', 5): 'Reine',
    ...
}
```

1.9.2 Question 1 : écrire une fonction `init_echiquier` qui initialise l'échiquier

Grâce à l'utilisation de boucles, créer dynamiquement un dictionnaire qui permettra de représenter l'ensemble des cases. Chaque case étant vide au départ, on lui affectera la valeur `None`.

```
In [ ]: def init_echiquier():
            lettres = ('a', 'b', 'c' ,'d', 'e', 'f', 'g', 'h')
            chiffres = (1, 2, 3, 4, 5, 6, 7, 8)
            echiquier = {}

            for lettre in lettres:
                for chiffre in chiffres:
                    echiquier[(lettre, chiffre)] = None

            return echiquier

        ## PP
        echiquier1 = init_echiquier()
        print(echiquier1)
```

1.9.3 Question 2 : ajout de pièces dans les cases

Dans le programme principal:

- Ajouter manuellement quelques pièces dans des cases de votre choix.
- Vérifier le contenu de votre dictionnaire.

```
In [ ]: ## PP
        echiquier1 = init_echiquier()
        echiquier1[('a', 1)] = 'Roi'
        echiquier1[('d', 5)] = 'Reine'
        print(echiquier1)
```

1.9.4 Question 3 : liste des cases occupées

- Ecrire une fonction `liste_cases_occupees` qui prend en paramètre un dictionnaire représentant un échiquier et qui renvoie en retour une liste représentant la liste des cases occupées.
- Dans le programme principal, afficher cette liste.

```
In [ ]: def liste_cases_occupees(echiquier):
            cases_occupees = []
            for case, piece in echiquier.items():
                if piece is not None:
                    cases_occupees.append(case)

            return cases_occupees

        ## PP
        echiquier1 = init_echiquier()
        echiquier1[('a', 1)] = 'Roi'
        echiquier1[('d', 5)] = 'Reine'
        print(liste_cases_occupees(echiquier1))
```