**Dictionnaires**

**Définition**

Le dictionnaire est un type de données fourni par Python, qui est une autre collection d'objets.  
Un dictionnaire est similaire à une liste à bien des égards :

* Il est mutable (ses éléments peuvent être modifiés)
* C'est dynamique (il peut grandir selon les besoins)

Les dictionnaires sont des collections non ordonnées d'éléments stockant des valeurs selon une paire clé-valeur.  
Avec un dictionnaire, vous créez des données structurées et récupérez des valeurs à l'aide d'une clé.  
On peut les comparer à un tableau de feuille de calcul : ils contiennent des colonnes avec des noms (clés) et des cellules avec des données (valeurs).

**Différence entre les dictionnaires et les listes**

Comment accéder aux éléments :  
Dans les listes, les éléments sont accessibles par leur position dans la liste (via l'indexation)  
Exemple :

my\_list = ['Rick', 'Sanchez']

print("My last name is:", my\_list[1])

Pour accéder au mot « Sanchez », il faut mettre son index entre crochets [].

Dans les dictionnaires, c'est un peu différent : les éléments n'ont pas d'index ; on y accède par des clés. Un dictionnaire est un tableau associatif, constitué d'une collection de paires clé-valeur. Chaque paire clé=valeur associe la clé à sa valeur associée.  
Par exemple, la clé first\_nameest associée à Rick, et la clé last\_nameest associée à Sanchez.

**Définir un dictionnaire**

Alors que les listes sont définies par des crochets [], les dictionnaires sont définis en encadrant une liste de paires clé-valeur entre accolades {}.  
Le signe deux-points : sépare chaque clé de sa valeur associée :

mon\_dict = {  
:,  
:,  
:  
}

rick\_dict = {

'first\_name':'Rick',

'last\_name':'Sanchez'

}

Les clés doivent être déclarées sous forme de chaînes entre guillemets. Les valeurs peuvent être de tout type.

my\_dog = {

'name': 'Rufus',

'age': 4,

'good\_dog': True

}

**Accéder aux données**

Pour accéder aux valeurs d'un dictionnaire, il suffit de référencer la clé de cette valeur entre parenthèses. C'est similaire à l'accès aux valeurs d'une liste, mais au lieu d'utiliser l'index de la valeur, on utilise le nom de sa clé.

my\_dog = {

'name': 'Rufus',

'age': 4

}

print(my\_dog['name'])

>> 'Rufus'

Exemple:

rick\_dict = {

'first\_name':'Rick',

'last\_name':'Sanchez'

}

print("The last name of rick is:", rick\_dict['last\_name'])

Si vous faites référence à une clé qui n’est pas dans le dictionnaire, vous obtiendrez une erreur :

print("The last name of rick is:", rick\_dict['age'])

Traceback (dernier appel le plus récent) :  
Fichier «« , ligne 1, dans  
rick\_dict['age']  
KeyError : 'âge'

Vous pouvez utiliser le déballage de tuples pour parcourir les clés et les valeurs du dictionnaire utilisé.  
Pour ce faire, il suffit de décompresser les éléments de chaque élément dans deux variables distinctes représentant la clé et la valeur :

a\_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}

print(a\_dict.items())

# output :

dict\_items([('color', 'blue'), ('fruit', 'apple'), ('pet', 'dog')])

# The items() method returns a view object that displays

# a list of dictionary's (key, value) tuple pairs.

for key, value in a\_dict.items():

print(key, '->', value)

# output

color -> blue

fruit -> apple

pet -> dog

**Types de données**

Vous pouvez stocker tout type de données dans un dictionnaire. Il peut s'agir de chaînes de caractères et d'entiers, mais aussi d'autres dictionnaires ou listes.

my\_dog = {

'name': 'Rufus',

'age': 4,

'best\_friend': {

'name': 'Felix',

'age': 4.5

},

'favorite\_foods': ['steaks', 'sausages', 'shawarma']

}

**Dictionnaires et listes**

Les dictionnaires peuvent être particulièrement utiles lorsqu'ils sont stockés dans une liste pour représenter une collection de plusieurs éléments représentant la même chose, comme une liste de produits.

shirts = [

{

'name': 'Awesome T-shirt 3000',

'size': 'S',

'price': 20

},

{

'name': 'Awesome T-shirt 3000',

'size': 'M',

'price': 25

},

{

'name': 'Awesome T-shirt 3000',

'size': 'L',

'price': 30

},

]

**🚀 Défi**

Accéder à la valeur de l'historique des clés

sample\_dict = {

"class":{

"student":{

"name":"Mike",

"marks":{

"physics":70,

"history":80

}

}

}

}

**Opérations sur les dictionnaires**

* Modifier une entrée dans un dictionnaire : Modifier un élément dans un dictionnaire fonctionne de la même manière qu'une liste, il faut d'abord le sélectionner (avec sa touche) puis lui attribuer une nouvelle valeur avec le signe égal =.

rick\_dict['last\_name'] = 'SANCHEZ'

* Ajout d'une entrée à un dictionnaire existant : Pour ajouter une entrée à un dictionnaire existant, il suffit d'attribuer une nouvelle clé à sa valeur, comme vous l'auriez modifiée s'il s'agissait d'une clé existante.

rick\_dict['hair\_color'] = 'white'

* Supprimer une entrée dans un dictionnaire : Pour supprimer une entrée, utilisez l'instruction del, en spécifiant la clé à supprimer :

del rick\_dict['hair\_color']

**Restrictions des clés**

Presque tous les types de valeurs peuvent être utilisés comme clé de dictionnaire en Python. Cependant, il existe quelques restrictions.

* Premièrement, les clés sont uniques ; elles ne peuvent apparaître qu'une seule fois dans un dictionnaire ; si vous attribuez une valeur à une clé de dictionnaire existante, la clé n'est pas ajoutée une deuxième fois mais remplace la valeur actuelle.
* Deuxièmement, la clé doit être de type immuable ; elle peut être un entier, un nombre à virgule flottante, une chaîne, un booléen et même un tuple, car tous ces types sont immuables. Une liste ne peut pas être une clé de dictionnaire.  
  En revanche, il n'y a aucune restriction sur les valeurs de dictionnaire.

**Le inmot clé**

Dans une liste, le inmot-clé renvoie Vrai ou Faux selon que l'opérande spécifié apparaît dans la liste.  
Un dictionnaire renvoie Vrai si l'opérande apparaît comme clé dans le dictionnaire.

**Itération**

L'itération sur une liste de dictionnaires est simple ; votre itérateur contient le dictionnaire que vous souhaitez consulter.

for shirt in shirts:

print(shirt['size'])

>> S

>> M

>> L

**Méthodes intégrées**

* keys() : La méthode my\_dict.keys() renvoie un dict\_keys de toutes les clés de my\_dict
* values() : La méthode my\_dict.values() renvoie un dict\_values ​​de toutes les valeurs de my\_dict
* items() : La méthode my\_dict.items() renvoie un dict\_items de tuples contenant les paires clé-valeur dans un dictionnaire.

rick\_dict = {

'first\_name':'Rick',

'last\_name':'Sanchez'

}

print(rick\_dict.items())

* mise à jour() : Siet d sont des dictionnaires, d.update() fusionne les entrées deen d.  
  Pour chaque clé dans:
* Si la clé n'est pas présente dans d, la paire clé-valeur deest ajouté à d.
* Si la clé est déjà présente dans d, la valeur correspondante dans d pour cette clé est mise à jour avec la valeur de.

**🚀 Défi**

Supprimer l'ensemble de clés du dictionnaire Python

sample\_dict = {

"name": "Kelly",

"age":25,

"salary": 8000,

"city": "New york"

}

keys\_to\_remove = ["name", "salary"]

**Conclusion**

Les dictionnaires sont un outil puissant en Python qui offre des capacités efficaces de manipulation des données. Ils sont particulièrement utiles pour l'analyse de données, le développement web et toutes les applications nécessitant une recherche rapide de valeurs de configuration ou de propriétés. La maîtrise des dictionnaires améliore votre capacité à travailler avec de grands ensembles de données et à développer des applications complexes pilotées par les données.