4 – Les dictionnaires

https://www.lumni.fr/video/les-dictionnaires#containerType=serie&containerSlug=la-maison-lumni-lycee

Définition:

Les éléments d'une liste sont repérés par des indices 0, 1, 2, ... Une différence essentielle avec un dictionnaire, objet de type dict, est que les indices sont remplacés par des objets du type str, float, tuple (si les n_uplets ne contiennent que des entiers, des flottants ou des p-uplets). On les appelle des clés et à chaque clé correspond une valeur. Ces clés ne sont pas ordonnées.

Création:

Les éléments d'un dictionnaire sont des couples clé-valeur. Un dictionnaire est créé avec des accolades, les différents couples étant séparés par des virgules. La clé et la valeur correspondante d'un élément sont séparées par deux-points.

```
dico1 = {'nom': 'tifaine', 'age': 33, 'status': 'enseignant'}
```

Les clés peuvent être de n'importe quel type (int, str, p-uplet, booléen) non-mutable.

Un dictionnaire est un conteneur (comme les listes ou les tuples), mutable.

Dans l'exemple précédent, les clés sont, respectivement, 'nom', 'age' et 'status', et les valeurs sont, respectivement, 'tifaine', 33 et 'enseignant'.

Exo:

Ecrire l'instruction suivante:

```
d = {chr(65+i): i for i in range(26)}
```

Afficher d

Il est possible de convertir une liste de listes en dictionnaire grâce à la fonction *dict()*

Ecrire les instructions suivantes:

```
liste = [['A', 0], ['B', 1], ['C', 2]]
d1 = dict(liste)
```

Afficher d1

Accéder aux éléments d'un dictionnaire:

Contrairement aux tuples ou aux listes, les dictionnaires ne sont pas indéxé, c'est à dire que l'on ne peux pas accéder à ses élément via un indice, mais via la clé associé à la valeur. Ainsi, pour accéder à l'élément 'tifaine' de notre dictionnaire, on pourrait procéder ainsi:

```
print(dico1['nom']) -> tifaine
```

C'est pour cette raison que toutes les clés d'un dictionnaire doivent être différentes.
Si l'on reprend notre dictionnaire d1:

Ecrire l'instruction: d1.keys() puis afficher le résultat, qu'obtenez-vous ?
-> dict_keys(['A', 'B', 'C'])

Ecrire l'instruction: d1.values() puis afficher le résultat, qu'obtenez-vous ?
-> dict_values([0, 1, 2])

Pour accéder à l'ensemble des couples clés/valeurs, nous utilisons la méthode *items()*

Ecrire l'instruction: d1.items(), puis afficher le résultat, qu'obtenez-vous ?
-> dict_items([('A', 0), ('B', 1), ('C', 2)])

Les couples clés/valeurs obtenus semble être d'un type familier, quel type ?
-> type tuple

Il est possible de tester l'appartenance à un dictionnaire avec le mot clé *in*:

```
"A" in d1        # teste si "A" est une clé
3 in d1.values()        # teste si 3 est une valeur
("C", 2) in d1.items    # teste si ("C", 2) est un couple clé/valeur
```

On en déduit 3 manières de parcourir un dictionnaire, lesquelles ?

- Via la clé
- Via la valeur
- Via le couple clé/valeur

Modification d'un dictionnaire:

Il est possible de supprimer une entrée de dictionnaire via le mot-clé 'del', exemple:

```
del dico1['nom']
```

Cette instruction va supprimer le couple clé/valeur du dictionnaire, ainsi, tenter d'accéder à cette clé va produire une erreur:

```
Traceback (most recent call last):
  File "...", line 7, in <module>
    print(dico1['nom'])
KeyError: 'nom'
```

Il est possible de modifier la valeur la valeur d'une clé:

```
dico1['age'] = 32
```

Ainsi, pour vérifier que la modification à bien été prise en compte:

```
print(dico1['age']) -> 32
```

Si, lors de la modification, le couple clé/valeur n'existe pas, il sera automatiquement créé:
```
dico1['prénom'] = 'tifaine'

print(dico1['prénom']) -> tifaine
```

Si l'on reprend notre dictionnaire d1:

Ecrire l'instruction suivante: d1["A"], puis afficher sa valeur, qu'obtenez-vous ?
-> 0

Ecrire l'instruction suivante: d1["A"] = 1, puis afficher sa valeur, qu'obtenez-vous ?
-> 1

Ecrire l'instruction suivante: d1["D"] = 3, puis afficher sa valeur, qu'obtenez-vous ?
-> un nouveau couple clé/valeur qui associe la clé "D" à la valeur 3

Ecrire l'instruction suivante: v = d1["E"], qu'obtenez-vous ?
-> une erreur car la clé "E" n'existe pas

La méthode *get()* permet de gérer ce genre de problème:
v = d1.get("A")
afficher v
-> 1
v = d1.get("E")
afficher v
-> None

La fonction *len()* renvoie le nombre d'éléments d'un dictionnaire:
print(len(d1)) -> 4

Itérer sur un dictionnaire:

Il est possible d'itérer sur un dictionnaire:

```
for (cle, val) in dico2.items():
    print(cle, ' -> ', val)

for elem in dico2.keys():       # Préciser le .keys() n'est pas obligatoire
    print(elem)

for elem in dico2.values():
    print(elem)
```

Dictionnaire en compréhension:

```
print({k: k**3 for k in range(2, 11)})#Il suffit de préciser la clé et la valeur (chaque élément)
-> {2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343, 8: 512, 9: 729, 10: 1000}
```

Copie:

Les comportements sont similaires à ceux rencontrés avec les listes, en particuliers si les valeurs sont des listes. Il est donc conseillé d'utiliser la fonction *deepcopy* du module *copy* pour être certain d'obtenir une "vraie" copie.

Les éléments d'un dictionnaire peuvent aussi être des dictionnaires:

```
vols = {'Lisbonne': {'heure': 21:10, 'num': 'EJU7674', 'compagnie': 'EASYJET'},
        'Vienne': {'heure': 21:25, 'num': 'OS430', 'compagnie': 'AUSTRIAN AIRLINES'},
        'Londres': {'heure': 21:55, 'num', 'BA357' 'compagnie': 'BRITISH AIRWAYS'} ...}

vols['Lisbonne']
-> {'heure': 21:10, 'num': 'EJU7674', 'compagnie': 'EASYJET'}
```

L'implémentation d'un dictionnaire optimise le coût en temps de la recherche d'un élément !

Application:

Les photos prises avec un appareil numérique contiennent de nombreuses informations. Dans le fichier image, par exemple au format jpeg, sont stockées des données non seulement sur l'image elle-même mais aussi sur l'appareil, le logiciel utilisé, et en particulier des données EXIF (Exchangeable Image File Format). Une partie est accessible dans les propriétés de fichier ou avec un logiciel de traitement d'images.

Les spécifications sont gérées par un organisme japonais, le JEITA, qui définit différents dictionnaires de référence permettant d'accéder à ces données. Les clés (les tags) et les valeurs sont des nombres écrits dans l'entête du fichier. Les dictionnaires donnent l'interprétation de ces clés. Par exemple la clé 256 a pour valeur 'Width', la largeur de l'image en pixel, la clé 257 a pour valeur 'Length', la hauteur de l'image en pixel.

Exo:

Se renseigner sur les différentes opérations et méthodes disponible pour les dictionnaires et créer un tableau (excel ou word) de la forme:

type opération / méthode | exemple concret | résultat exemple

créer un dictionnaire | dictionnaire_vide = [] | print(dictionnaire_vide) -> {}
créer un dictionnaire | dictionnaire_int = [1: 2] | print(dictionnaire_int) -> {1: 2}
etc...
méthode .keys() | dictionnaire_int.keys() | print(dictionnaire_int.keys()) -> dict_keys([1])
etc...

Créer un dictionnaire vide: dico = ...

Dans le dictionnaire créé, ajouter la clé 'bit' associé à la chaîne de caractères suivante:

"élément d'information valant 0 ou 1"

Associer "processeur" à "unité de calcul de l'ordinateur"

Saisir l'instruction suivante, que remarquez-vous ? Proposer une explication.

print(dico[0])

Quelle est l'instruction à écrire pour afficher la définition du mot 'bit' ?

Ecrire l'instruction pour relier les mots suivant a leurs verbes irrégulier en Anglais:
Appeler le dictionnaire " mots"

Avoir -> have, had, had
Savoir -> know, knew, known
Aller -> go, went, gone
Prendre -> take, took, taken
Chercher -> seek, sought, sought

Les méthodes suivantes permettent d'intérrooger le dictionnaire, donner leur définition:

.items() ->
.keys() ->
.values() ->
lent(dico) ->

Que ce passe-t-il si l'on tente d'insérer 2 entrées ayant la même clé ? Tester cette éventualité et proposer une explication:

Ecrire une fonction *cherche(verbe, dictionnaire)* qui renvoie True si le verbe cherché est dans le dictionnaire, sinon, False:

Ecrire l'instruction Python suivante, tester la, expliquer:

cherche('avoir', mots)