

Les dictionnaires

Les dictionnaires sont des collections similaires aux listes, mais au lieu d'utiliser des index (0, 1, 2, etc...), on utilise des clés alphanumériques ("Poirés", "Pommes", "Fraises", etc...).

Les clés sont non ordonnées (c'est-à-dire qu'elles ; ne sont pas forcément rangées dans le même ordre), ainsi lors de l'affichage d'un dictionnaire l'ordre des éléments peut différer d'une exécution à l'autre.

Il faut penser à ne jamais écrire un programme qui prend en compte l'ordre des éléments d'un dictionnaire.

Clés (keys)	Poirés	Pommes	Fraises	Éléments (Items)
Valeurs (values)	5	10	35	

Q1 Donner le résultat des exemples suivants

	Instructions	Résultat affiché
Affichage valeurs	<pre>stock={'poires':5, 'pommes':10, 'fraises':35} print(len(stock)) print(stock['fraises'])</pre>	<p>3</p> <p>35</p>
Affichage clés	<pre>stock={'poires':5, 'pommes':10, 'fraises':35} inventaire="" for nom in stock.keys(): inventaire += nom + ', ' print(inventaire)</pre>	<p>poires, pommes, fraises,</p>
Affichage valeurs	<pre>stock={'poires':5, 'pommes':10, 'fraises':35} for num in stock.values(): print (num)</pre>	<p>35</p>

Affichage des clés et valeurs	<pre>stock = {'poires': 5, 'pommes': 10, 'fraises': 35} for nom, num in stock.items(): print(nom, '->', num)</pre>	fraises -> 35
Opérations booléennes	<pre>stock = {'poires': 5, 'pommes': 10, 'fraises': 35} print('fraises' in stock.keys()) print('banane' in stock.keys()) print(7 in stock.values()) print(5 in stock.values()) print(3 in stock.values())</pre>	<p>True False False True False</p>
Ajout et édition d' éléments	<pre>stock = {'poires': 5, 'pommes': 10, 'fraises': 35} print(stock) stock["fraises"]=20 print(stock) stock["abricots"]=15 print(stock)</pre>	<pre>{'poires': 5, 'pommes': 10, 'fraises': 35} {'poires': 5, 'pommes': 10, 'fraises': 20} {'poires': 5, 'pommes': 10, 'fraises': 20, 'abricots': 15}</pre>

Suppression d' éléments	<pre>stock = {'poires': 5, 'pommes': 10, 'fraises': 35} print(stock) del stock["fraises"] print(stock) valeur = stock.pop("pommes") print(stock) print(valeur)</pre>	<pre>{'poires': 5, 'pommes': 10, 'fraises': 35} {'poires': 5, 'pommes': 10} {'poires': 5} 10</pre>
-------------------------	---	--

Exercice 1 : Entraînement a la manipulation des dictionnaires

Soit le dictionnaire suivant :

```
d = { 'nom': 'Dupuis', 'prenom': 'Jacques', 'age': 30 }
```

Q2 Donner le code python qui répond aux questions suivantes

Corriger l'erreur dans le prénom, la bonne valeur est 'Jacques'.
<pre>d={'nom': 'Dupuis', 'prenom': 'Jacques', 'age': 30},</pre>
Afficher la liste des clés du dictionnaire.
<pre>print(liste(d.keys()))</pre>
Afficher la liste des valeurs du dictionnaire.
<pre>print(list(d.values()))</pre>
Afficher la liste des paires clé/valeur du dictionnaire.
<pre>for cle, valeur in d.items(): print</pre>
Ecrire la phrase "Jacques Dupuis a 30 ans."
<pre>print(d['prenom'], d['nom'], " a ", d['age'], " ans")</pre>

Exercice 2 : Un dictionnaire en compréhension

Q3 écrire une instruction python qui construit le dictionnaire D suivant, à l'aide d'une définition par compréhension :

```
D = {'0': 0, '1': 0, '2': 0, '3': 0, '4': 0, '5': 0, '6': 0, '7': 0, '8': 0, '9': 0}
D = {str(i): 0 for i in range(10)}
```

Q4 Dire ce que fait la fonction suivante :

```
def comptage(texte):
    dic={ element : texte.count(element) for element in texte}
    return dic
```

compte le nombre de fois que chaque element qui est entré dans texte et ça retourne un dico ou les clés

Exercice 4 : Anagramme

Définition d'une anagramme :

Mot formé en changeant de place les lettres d'un autre mot.

Exemple : Chien, chine, niche.

Q5 En utilisant la fonction comptage de la question Q6, réaliser une fonction booléenne ***anagramme(chaine1, chaine2)*** qui prend 2 mots en paramètre et renvoie True si *chaine2* est une anagramme de *chaine1*, sinon False.

Exemples	Résultat dans la console
<code>print(anagramme('tortue', 'tourne'))</code>	False
<code>print(anagramme('imaginer', 'migraine'))</code>	True

Code Python

```
def anagramme(chainel, chaine2) :
    #A compléter
```

Q6 La fonction anagramme fonctionne-t-elle avec l'exemple suivant ?

```
print(anagramme('le rechauffement climatique', 'ce fuel qui tache le firmament' ))
```

Q7 Pourtant cela est bien une anagramme. Identifier le problème.


Q8 Supprimer alors l'élément qui pose un problème avant d'effectuer le test des 2 dictionnaires. (Utiliser la fonction ***del***)

```
def anagramme(chainel, chaine2) :
    occurrenceChainel = comptage(chainel)
    occurrenceChaine2 = comptage(chaine2)
    # a compléter
```

Exercice 5 : Rendu de monnaie

On veut réaliser une fonction **renduMonnaie(somme,pieces)** qui détermine les pièces à rendre dans un monnayeur.

Exemple :

Rendre la somme de 8€	Solution
	<p>1 billet de 5€ 1 pièce de 2€ 1 pièce de 1€</p>

Q9 Traduire l'algorithme en code python

Algorithme pseudo code	Python
<p>Fonction renduMonnaie (somme en entier, pièces : liste des pièces du monnayeur dans l'ordre décroissant) : dictionnaire des pièces choisies</p> <p> initialiser à zéro le dictionnaire choisies</p> <p> Pour p dans pieces</p> <p> choisies[p] ← 0</p> <p> Tant que somme ≥ p</p> <p> somme ← somme - p</p> <p> choisies[p] ← choisies[p] + 1</p> <p> fin tant que</p> <p> fin pour</p> <p> retourner choisies</p>	<pre>def renduMonnaie(somme, pieces) : choisies = {} # A compléter return choisies</pre>

Résultat dans la console	
<pre>#pieces en centimes d'euros pieces=[500,200,100,50,20,10,5,2,1] somme=780 print('Les pièces choisies sont') print(renduMonnaie(somme,pieces))</pre>	<p>Les pièces choisies sont</p> <p>{500: 1, 200: 1, 100: 0, 50: 1, 20: 1, 10: 1, 5: 0, 2: 0, 1: 0}</p>

Autre solution

Algorithme pseudo code	Python
Fonction renduMonnaie (somme en entier, pièces : liste des pièces du monnayeur dans l'ordre décroissant) : dictionnaire des pièces choisies initialiser à zéro le dictionnaire choisie Pour p dans pieces nb ← somme division entière par p choisies[p] ← nb somme ← somme – nb * p fin pour retourner choisies	<pre>def renduMonnaie(somme, pieces): choisies = {} for p in pieces: #A compléter return choisies</pre>

Rappel : La division entière (on ne garde pas les nombres après la virgule) se fait avec 2 barres obliques //

Annexe

Création d'un dictionnaire <i>d</i>	<code>d = {}</code> <code>d = dict()</code> <code>d = {k1:v1, k2:v2, ...}</code>
Accès à la valeur de la clé <i>key</i>	<code>value = d[key]</code>
Ajout et édition de la valeur de la clé <i>key</i>	<code>d[key] = value</code>
Test d'appartenance de la clé <i>key</i> dans <i>d</i>	<code>If key in d :</code>
Boucle pour sur chaque clé <i>key</i> de <i>d</i>	<code>For key in d :</code>
Tests d'égalité	<code>d1==d2</code> <code>d1 !=d2</code>
Suppression d'un item dont la clé est <i>key</i>	<code>del(d[key])</code> <code>del d[key]</code>

Manipulation des dictionnaires :

Supprime tous les éléments le <i>d</i>	<code>d.clear()</code>
Crée une copie de <i>d</i>	<code>d.copy()</code>
Crée un dictionnaire à partir d'une liste de clés <i>ks</i> et une valeur commune <i>v</i>	<code>ks = [k1, k2, ...]</code> <code>{}.fromkeys(ks, v)</code>
Liste les éléments de <i>d</i>	<code>d.items()</code>
Liste les clés de <i>d</i>	<code>d.keys()</code>
Liste les valeurs de <i>d</i>	<code>d.values()</code>
Renvoie la valeur de la clé <i>key</i> si elle existe, sinon renvoie <i>value</i>	<code>d.get(key [, value])</code>
Retire l'item de la clé <i>key</i> et renvoie sa valeur	<code>d.pop(key)</code>
Renvoie <i>value</i> si <i>key</i> n'est pas une clé de <i>d</i>	<code>d.pop(key [, value])</code>
Renvoie la valeur de la clé <i>key</i> si elle existe, sinon renvoie <i>value</i> et crée <i>d[key] = value</i>	<code>d.setdefault(key [, value])</code>
Ajoute à <i>d</i> la liste de tuples <i>ts</i>	<code>d.update(ts)</code>