TP Chapitre 4 : Dictionnaires

Objectifs du TP

Objectifs du TP. À l'issue de cette séance, tu dois être capable de :

- créer un dictionnaire (accolades ou `dict()`) et accéder à une valeur par sa clé ;
- ajouter, modifier et supprimer des couples clé-valeur (`dico[clé] = valeur`, `pop`) ;
- tester la présence d'une clé avec `in` et éviter les erreurs `KeyError` grâce à `get` ;
- parcourir un dictionnaire selon ses clés, ses valeurs ou ses couples (`keys()`, `values()`, `items()`) ;
- construire un dictionnaire au fil d'une boucle (accumulation par clé) ;
- concevoir et écrire un programme complet à partir d'une spécification (entrées, traitement, sorties).

Consignes générales. Tous les exercices se traitent en Python (console ou fichier). Enregistre régulièrement ton travail. Quand un programme ne fonctionne pas, lis le message d'erreur : il indique souvent la ligne et le type d'erreur. Rappel du cours : un dictionnaire n'est pas ordonné, ne suppose jamais de « premier » ou de « dernier » élément dans tes affichages. La partie B se fait *en autonomie* : tu écris des programmes entiers à partir d'une spécification, aucune ligne de code n'est fournie.

Partie A — Application guidée

Les exercices suivants se programment en Python. Ils sont classés du plus simple au plus difficile. Chaque exercice se termine par un *point de validation* (encadré vert) : le résultat exact que doit produire ton programme pour être juste. Complète les squelettes de code en respectant l'indentation.

Exercice A1 — Mon budget mensuel (création d'un dictionnaire et accès par clé)

Tu gères ton budget mensuel : chaque poste de dépense (loyer, courses, transport, loisirs) est associé à son montant en euros. Le dictionnaire `depenses` regroupe ces quatre postes. Complète le programme : il doit afficher le montant du loyer, puis le nombre de postes de dépense.

```
depenses = {"loyer": 650, "courses": 180, "transport": 60, "loisirs": 40}

# A COMPLETER : afficher le montant du loyer avec la notation depenses["loyer"]

# A COMPLETER : afficher le nombre de postes de depense avec len
```

Point de validation

Le programme doit afficher `650` (le loyer), puis `4` (le nombre de postes de dépense).

Exercice A2 — La boutique de vêtements (ajout, modification, suppression)

Une boutique de vêtements tient un catalogue associant chaque article à son prix en euros. Complète le programme :

1. ajouter la veste au catalogue, au prix de 85 € ;
2. appliquer la promotion : le jean passe à 35 € ;
3. retirer les baskets du catalogue avec `pop`, en affichant la valeur retirée ;

4. afficher le catalogue.

```
catalogue = {"t-shirt": 15, "jean": 40, "baskets": 60}

# A COMPLETER : ajouter la veste au catalogue, au prix de 85 euros

# A COMPLETER : le prix du jean passe a 35 euros (promotion)

# A COMPLETER : retirer les baskets du catalogue avec pop
# et afficher la valeur retiree

print(catalogue)
```

Point de validation

Le `print` de la valeur retirée doit afficher `60` ; le dictionnaire final doit contenir exactement les couples `{"t-shirt": 15, "jean": 35, "veste": 85}`.

Exercice A3 — Le panier de courses (parcours et total)

Ton panier de courses contient quatre articles avec leur prix en euros. Complète le programme : parcours le dictionnaire avec une boucle `for` et additionne tous les prix pour obtenir le total du panier.

```
panier = {"pain": 1.2, "lait": 0.95, "oeufs": 2.4, "fromage": 3.5}
total = 0

# A COMPLETER : boucle for sur les cles du panier
# ajouter chaque prix au total

print("Total du panier :", total, "euros")
```

Point de validation

Le programme doit afficher environ `Total du panier : 8.05 euros` (un léger écart sur les décimales est normal, c'est la représentation des flottants).

Exercice A4 — La boutique en ligne (tester une clé avec `in`)

Le site de la boutique affiche le prix d'un produit demandé. Complète le programme : si le produit est dans le catalogue, affiche son prix ; sinon affiche `Produit inconnu`, sans jamais lever d'erreur.

```
catalogue = {"t-shirt": 15, "jean": 40, "baskets": 60}
produit = input("Nom du produit : ")

if produit in catalogue:
    # A COMPLETER : afficher le prix du produit
else:
    # A COMPLETER : afficher "Produit inconnu"
```

Point de validation

Avec `jean`, le programme doit afficher `Prix : 40 euros` ; avec `chapeau`, il doit afficher `Produit inconnu`.

Partie B — En autonomie

Tu écris maintenant des programmes *entiers*, sans squelette. Pour chaque exercice : lis la spécification (entrées, traitement, sorties, contraintes), réponds d'abord aux questions de conception de l'encadré orange (sur papier ou à voix haute), puis écris ton programme dans un fichier et teste-le avec les valeurs de l'encadré vert. L'objectif est de concevoir toi-même la solution : ne demande pas de lignes de code toutes faites.

Exercice B1 — La caisse : recettes par catégorie (accumulation par clé)

Un petit commerce enregistre chaque vente avec sa catégorie (par exemple `"alimentation"`, `"vetements"`, `"loisirs"`) et son montant en euros. Le gérant veut connaître le total des ventes réalisées dans chaque catégorie : c'est une association catégorie → total, donc un dictionnaire.

Énoncé. Écrire un programme qui demande le nombre de ventes, puis, pour chaque vente, sa catégorie et son montant. Le programme construit un dictionnaire associant à chaque catégorie le total des montants de cette catégorie, puis affiche chaque catégorie avec son total.

Spécification.

— *Entrées* : un entier N (nombre de ventes, $N \geq 1$), puis N paires (catégorie : texte, montant : flottant en euros).

— *Traitement* : partir d'un dictionnaire vide ; pour chaque vente, ajouter le montant au total de sa catégorie — si la catégorie n'existe pas encore, l'initialiser puis ajouter.

— *Sorties* : pour chaque catégorie, une ligne `categorie : total euros`.

— *Contraintes* : boucle `for` ; les ventes ne sont pas mémorisées (on ajoute le montant au fur et à mesure) ; la méthode `get` est obligatoire pour l'accumulation.

Pour t'aider — questions de conception (sans donner le code)

— Pourquoi un dictionnaire est-il plus adapté qu'une liste pour regrouper des montants par catégorie ?

— Que renvoie `totaux.get("vetements", 0)` quand `"vetements"` n'est pas encore une clé ? Et quand elle en est une ?

— Pourquoi écrire `totaux[categorie] = totaux.get(categorie, 0) + montant` et pas `totaux[categorie] = montant` ?

— Quelle erreur lèverait `totaux[categorie] = totaux[categorie] + montant` pour une catégorie jamais rencontrée ?

Point de validation

Avec $N = 4$ et les ventes : `alimentation` 12,5 ; `vetements` 30 ; `alimentation` 7,5 ; `loisirs` 20 — le programme doit afficher les trois lignes `alimentation : 20.0 euros`, `vetements : 30.0 euros` et `loisirs : 20.0 euros` (l'ordre d'affichage n'est pas garanti).

Exercice B2 — Le portefeuille d'actions (dictionnaire imbriqué)

Tu possèdes un petit portefeuille d'actions. Pour chaque action, on connaît le nombre de titres possédés et le prix d'achat unitaire. Le dictionnaire `portefeuille` contient ces informations : à chaque action (clé) est associé un dictionnaire avec les clés `"quantite"` et `"prix"`.

```
portefeuille = {"TechCorp": {"quantite": 10, "prix": 45.5},
                "EcoBank": {"quantite": 25, "prix": 12.0},
                "Solaria": {"quantite": 5, "prix": 80.0}}
```

Énoncé. Écrire un programme qui calcule la valeur de chaque action (quantité × prix) et la valeur totale du portefeuille, puis affiche une ligne par action et le total.

Spécification.

— *Entrées* : aucune saisie ; le dictionnaire `portefeuille` est fourni ci-dessus.

— *Traitement* : parcourir le portefeuille avec `items()` ; pour chaque action, `valeur = quantite * prix` ; cumuler dans un total.

— *Sorties* : une ligne par action `Action : valeur euros`, puis `Total du portefeuille : ... euros`.

— *Contraintes* : dictionnaire imbriqué ; parcours avec `items()` obligatoire ; pas de saisie clavier.

Pour t'aider — questions de conception (sans donner le code)

— Comment accéder au nombre de titres de TechCorp ? (indiquer les deux clés à enchaîner)

— Pourquoi `items()` est-il plus pratique que `keys()` ici ?

— Quelle variable cumule la valeur totale ? Où doit-elle être initialisée ?

— Que se passerait-il si l'ordre des lignes affichées changeait ? (réponds en une phrase)

Point de validation

Le programme doit afficher `TechCorp : 455.0 euros`, `EcoBank : 300.0 euros`, `Solaria : 400.0 euros`, puis `Total du portefeuille : 1155.0 euros`.

Exercice B3 — Les comptes bancaires (recherche et client le plus riche)

Une banque associe à chaque client le solde de son compte (en euros). Le directeur veut : afficher le solde d'un client demandé ; trouver le client au solde le plus élevé ; calculer le total des soldes de tous les clients.

Énoncé. Écrire un programme qui demande le nom d'un client, affiche son solde (ou un message si le client n'existe pas), puis affiche le nom du client le plus riche avec son solde, puis le total des soldes.

Spécification.

— *Entrées* : le nom d'un client (texte) ; le dictionnaire des soldes est fourni : `{"Lina": 1250.0, "Sami": 480.0, "Zoe": 2300.5, "Noe": 75.0}`.

— *Traitement* : tester la présence du client avec `in` ; parcourir le dictionnaire avec `items()` pour calculer le total et repérer le solde maximum.

— *Sorties* : `Solde de X : ... euros` ou `Client inconnu` ; `Client le plus riche : ... ... euros` ; `Total des soldes : ... euros`.

— *Contraintes* : opérateur `in` obligatoire pour la recherche ; parcours `items()` obligatoire ; fonction `max` interdite.

Pour t'aider — questions de conception (sans donner le code)

— Pourquoi un dictionnaire est-il naturel ici : à quoi servent les clés, à quoi servent les valeurs ?

— Comment repérer le client le plus riche sans fonction toute faite ? (une variable qui mémorise le meilleur trouvé jusqu'ici, comparée à chaque tour)

— Comment initialiser cette variable avant la boucle ?

— Pourquoi ne pas écrire directement `soldes[client]` pour afficher le solde demandé ? Que se passe-t-il si le client n'existe pas ?

Point de validation

Avec la demande `Sami : Solde de Sami : 480.0 euros`, `Client le plus riche : Zoe 2300.5 euros`, `Total des soldes : 4105.5 euros`. Avec la demande `Fanny : Client inconnu` (les deux autres lignes restent identiques).

Exercice B4 — La conversion de devises (`get` et valeur par défaut)

Un site de voyage affiche les prix en euros. Le voyageur choisit une devise ; le programme affiche le prix converti. Si le code de la devise n'est pas connu, le programme affiche un message d'erreur — sans jamais planter.

Énoncé. Écrire un programme qui demande un prix en euros et un code de devise (parmi `USD`, `GBP`, `JPY`, `CAD`), puis affiche le prix converti à l'aide du dictionnaire des taux, ou `Devise inconnue` si le code n'y figure pas.

Spécification.

— *Entrées* : un prix en euros (flottant), un code de devise (texte).

— *Traitement* : lire le taux dans `taux_devises` ; si le code est absent, message d'erreur ; sinon multiplier le prix par le taux.

— *Sorties* : `Prix : ... devise` (montant arrondi à 2 décimales) ou `Devise inconnue`.

— *Contraintes* : méthode `get` obligatoire ; aucun test avec `in` ; le programme ne doit jamais lever `KeyError`.

Pour t'aider — questions de conception (sans donner le code)

— Que renvoie `taux_devises.get("BRL")` si `"BRL"` n'est pas une clé ? Et `taux_devises.get("BRL", 0)` ?

— Comment distinguer, après un `get`, une devise valide d'une devise inconnue ?

— Pourquoi `get` est-il plus sûr que la notation `taux_devises[devise]` ?

— À quoi servirait un second dictionnaire associant à chaque code le symbole de la devise (`"USD"` → `"$"`, `"GBP"` → `"£"`, `"JPY"` → `"¥"`) ?

Point de validation

Prix 50 et devise `USD` → environ `Prix : 54.5 USD` ; prix 100 et devise `JPY` → `Prix : 16200.0 JPY` ; devise `BRL` → `Devise inconnue`.

Partie C — Exercice de réflexion

À finir à la maison

Mon budget mensuel : suivi des dépenses par catégorie.

Contexte. Tu tiens à jour ton budget du mois. Chaque dépense appartient à une catégorie (logement, alimentation, transport, loisirs, santé…). Tu t'es fixé un budget prévu par catégorie, et tu veux savoir, à la fin du mois : combien ai-je dépensé par catégorie ? combien au total ? quelle catégorie a coûté le plus ? où ai-je dépassé mon budget ?

Objectif. Écrire un programme complet qui :

— enregistre les dépenses du mois (saisie au clavier) ;

— les regroupe par catégorie dans un dictionnaire (accumulation par clé) ;

— affiche le total par catégorie et le total général ;

— identifie la catégorie la plus coûteuse ;

— compare chaque catégorie au budget prévu et affiche `depassement`, `reste` ou `budget respecte`.

Questions à se poser avant de coder.

— Quelles sont les données d'entrée ? (nombre de dépenses, puis pour chacune : catégorie et montant)

— Quel dictionnaire construire ? Que représentent les clés ? les valeurs ?

— Comment accumuler les montants par catégorie sans erreur `KeyError` ? (la méthode `get` est ton amie)

— Comment calculer le total général ? Comment trouver la catégorie la plus coûteuse ?

— Comment comparer au budget prévu, y compris pour une catégorie sans aucune dépense ? (revoir l'exercice B4)

— Comment vérifier que le programme est juste ? (petit exemple calculé à la main)

Étapes suggérées.

1. Version 0 : un dictionnaire de dépenses écrit en dur ; afficher le total général.
2. Version 1 : saisie des dépenses avec une boucle `for` ; accumulation par catégorie avec `get`.
3. Version 2 : affichage par catégorie, total général, catégorie la plus coûteuse (motif de l'exercice B3).
4. Version 3 : comparaison avec le budget prévu (deux dictionnaires) ; trois cas possibles par catégorie.
5. Bonus : afficher le pourcentage du budget consommé par catégorie ; tester avec un mois où une catégorie prévue n'est pas utilisée du tout.

Contraintes. Données crédibles (catégories réalistes : logement, alimentation, transport, loisirs, santé ; montants entre 1 € et 500 €) ; programme testé et commenté ; les messages peuvent être écrits sans accents (ex. `Categorie`, `depassement`) si ton éditeur pose problème ; ne jamais supposer d'ordre d'affichage d'un dictionnaire.

Fin du TP — la partie C est à finir à la maison.