TP : Exercices sur les Dictionnaires en Python

Exercice 1 : Création et accès

Crée un dictionnaire `personne` avec les clés `"nom"`, `"age"`, `"ville"`. Affiche la valeur associée à `"nom"` et à `"ville"`.

Exercice 2 : Ajout et modification

À partir de `personne`, ajoute la clé `"tel"` avec un numéro fictif, puis modifie `"ville"` par une autre ville. Affiche le dictionnaire mis à jour.

Exercice 3 : Suppression et accès sécurisé

Supprime la clé `"tel"` si elle existe. Affiche la note via `personne.get("note", "pas de note")`. Explique la différence entre `d["cle"]` et `d.get("cle")`.

Exercice 4 : Parcours de clés / valeurs

Crée un dictionnaire `etudiant = {"nom": "Alice", "note": 15, "groupe": "A"}`. Parcours et affiche : (1) uniquement les clés, (2) uniquement les valeurs, (3) les paires `clé : valeur` avec `items()`.

Exercice 5 : Compteur de fréquences (lettres)

Écris un programme qui compte le nombre d'occurrences de chaque lettre dans une chaîne donnée (ex. `"python est cool"`). Utilise `d.get(lettre, 0) + 1`.

Exercice 6 : Dictionnaire de listes

Crée `notes = {"math": [15, 12, 18], "info": [10, 14]}`. Ajoute une nouvelle note à `"info"`, puis calcule la moyenne des notes pour chaque matière et affiche `matiere -> moyenne`.

Exercice 7 : Liste de dictionnaires

Crée une liste `etudiants` contenant 3 dictionnaires {`"nom"`, `"note"`}. Affiche le nom et la note de chaque étudiant. Trouve l'étudiant avec la meilleure note (sans `max` au début, puis avec `max` et une `lambda`).

Exercice 8 : Fusion et mise à jour

Soient `a = {"x": 1, "y": 2}` et `b = {"y": 99, "z": 3}`. Crée un nouveau dictionnaire `c` qui fusionne `a` et `b` (les valeurs de `b` priment en cas de conflit). Montre deux méthodes : `{**a, **b}` et `a.copy(); a.update(b)`.

Exercice 9 : Inversion clé/valeur

Écris une fonction `inverser(d)` qui inverse un dictionnaire simple (`clé->valeur` devient `valeur->clé`). Explique et gère le cas où plusieurs clés partagent la même valeur (utilise alors une liste de clés).

Exercice 10 : Compréhensions de dict

À partir de la liste `noms = ["alice", "bob", "chloe"]`, construis un dict `{nom: len(nom)}` avec une compréhension de dictionnaire. Crée une deuxième version qui ne garde que les noms de longueur ≥ 4.

Exercice 11 : `setdefault` et regroupement

Écris un programme qui regroupe des prénoms par leur première lettre : `prenoms = ["Ali", "Amine", "Sara", "Samir", "Noa"]`
Résultat attendu (ex.) : `{"A": ["Ali","Amine"], "S": ["Sara","Samir"], "N": ["Noa"]}`
Astuce : `d.setdefault(cle, []).append(valeur)`.

Exercice 12 : Tri par valeurs

Soit `scores = {"alice": 15, "bob": 9, "chloe": 18, "dali": 12}`. Affiche les paires triées par score croissant puis décroissant. Montre une version qui retourne une `list` de tuples triés et une version qui reconstruit un dict ordonné (en Python 3.7+, l'ordre d'insertion est préservé).

Exercice 13 : Dictionnaires imbriqués

Crée `classe = {"A": {"alice": 15, "bob": 12}, "B": {"chloe": 18}}`. Ajoute un nouvel élève à la classe B, modifie la note de `"bob"` en classe A, puis calcule la moyenne par classe et la moyenne générale.

Exercice 14 : Validation de clés

Écris une fonction `verifier_cles(d, obligatoires)` qui vérifie que toutes les clés d'une liste `obligatoires` sont présentes dans `d`. Retourne `True/False` et liste les clés manquantes si nécessaire.

Exercice 15 : Comptage de mots (mini-texte)

Demande une phrase à l'utilisateur et construis un dict `mot->fréquence` (en séparant sur les espaces, en mettant en minuscules, et en retirant la ponctuation simple `.,;!?`). Affiche les 3 mots les plus fréquents.

Exercice 16 : Filtrage par condition

Soit `produits = {"pomme": 2.0, "banane": 1.0, "mangue": 3.5, "poire": 2.2}` (prix en €). Construis un nouveau dict ne contenant que les produits à moins de 2€ puis un autre à 2€ ou plus. Affiche le total de chaque catégorie.

Exercice 17 : Normalisation de données

Soit `bruts = {"A": 10, "B": 20, "C": 30}`. Transforme en `pourcentages` tels que la somme vaille 100 (arrondi à une décimale). Affiche le résultat sous la forme `"A: 16.7%"`.

Exercice 18 : Sécurisation d'accès

Écris une fonction `safe_get(d, chemin)` où `chemin` est une liste de clés pour naviguer dans un dict potentiellement imbriqué. Si une clé manque, retourne `None` sans lever d'exception. Exemple : `safe_get(cfg, ["db","host"])`.

Exercice 19 : Transformation de structure

Convertis la liste de tuples `[("alice",15),("bob",12),("chloe",18)]` en dict, puis reconvertis le dict en liste de tuples triée par nom, puis par note.

Exercice 20 : Mini-ORM (clé primaire)

Crée un dict `users` où chaque clé est un `id` unique (entier) et la valeur est un dict `{"nom", "email"}`. Ajoute une fonction `create_user(users, nom, email)` qui crée un nouvel `id` auto-incrémenté, une fonction `read_user(users, id)`, `update_user(users, id, **attrs)` et `delete_user(users, id)`.