Liste d'exercices pour les kholles de NSI

Exercice : Calculs mathématiques

Écrire une fonction `calculer_moyenne_ponderee(notes: list, coefficients: list) -> float` qui :

- Calcule la moyenne pondérée des notes selon leurs coefficients
- Vérifie que les listes ont la même taille
- Gère les erreurs possibles (division par zéro, listes vides)
- Calculer la compléxité de cet algorithme

Exercice : Manipulation de chaînes

Écrire une fonction `compter_voyelles(chaine: str) -> int` qui :

- Compte le nombre de voyelles dans une chaîne de caractères
- Considère les voyelles "a", "e", "i", "o", "u" (en minuscules)
- Exemple : compter_voyelles("bonjour") → 3
- Calculer la compléxité de cet algorithme

Exercice : Tuple vers liste

Écrire une fonction `tuple_vers_liste(tabeau: tuple) -> int` qui :

- Construit une liste contenant tous les éléments du tuple passé en paramètres.
- On evitera d'utiliser le cast `list(sequence)` qui permet de transformer une sequence quelconque en liste.
- Exemple : tuple_vers_liste((2,3,1,"bonjour')) → [2,3,1,"bonjour]
- Calculer la compléxité de cet algorithme.

Exercice : Recherche dans une liste

Écrire une fonction `recherche_elements(liste: list, element: any) -> list` qui :

- Renvoie la liste des indices où apparaît l'élément recherché
- Renvoie une liste vide si l'élément n'est pas trouvé
- Exemple : recherche_elements([1,2,1,3,1], 1) → [0,2,4]
- Calculer la compléxité de cet algorithme
- Expliquer la terminaison de cet algorithme

Exercice : Filtrage dans une liste

Écrire une fonction `filtrer_pairs(liste: list) -> list` qui :

- Renvoie une nouvelle liste contenant uniquement les nombres pairs de la liste d'origine
- Exemple : filtrer_pairs([1, 2, 3, 4, 5, 6]) → [2, 4, 6]
- Calculer la compléxité de cet algorithme
- Expliquer la terminaison de cet algorithme

Exercice : Dictionnaires

Écrire une fonction `inverser_dictionnaire(d: dict) -> dict` qui :

- Inverse les clés et les valeurs d'un dictionnaire
- Suppose que toutes les valeurs sont uniques
- Exemple : inverser_dictionnaire({'a': 1, 'b': 2}) → {1: 'a', 2: 'b'}
- Calculer la compléxité de cet algorithme

Exercice : Comptage d'occurrences

Écrire une fonction `compter_occurrences(liste: list) -> dict` qui :

- Compte le nombre d'occurrences de chaque élément dans une liste
- Renvoie un dictionnaire avec les éléments comme clés et les occurrences comme valeurs
- Exemple : compter_occurrences([1, 2, 2, 3, 3, 3]) → {1: 1, 2: 2, 3: 3}
- Calculer la compléxité de cet algorithme
- Expliquer la terminaison de cet algorithme

Exercice : Fusion de listes

Écrire une fonction `fusionner_listes(liste1: list, liste2: list) -> list` qui :

- Fusionne deux listes en une seule, sans doublons
- Renvoie la liste fusionnée
- Exemple : fusionner_listes([1, 2, 3], [3, 4, 5]) → [1, 2, 3, 4, 5]
- Calculer la compléxité de cet algorithme

Voici huit autres exercices dans le même esprit, qui permettront de travailler la programmation en Python (boucles, conditions, listes, dictionnaires, récursivité, etc.) ainsi que quelques notions d'analyse de la complexité et de terminaison.

Nouveaux Exercices Python pour NSI – Niveau Débutant

Exercice 1 : Inversion d'une liste

Objectif : Écrire une fonction qui renverse une liste sans utiliser la méthode intégrée `reverse()` ni le cast `[::-1]`.

Consignes de base

1. **Fonction à réaliser** :
   Écrire une fonction `inverser_liste(liste: list) -> list` qui :
   - Parcourt la liste initiale de la fin vers le début.
   - Construit et renvoie une nouvelle liste contenant les éléments dans l'ordre inverse.
   - Exemple :
     `inverser_liste([1, 2, 3, 4])` doit renvoyer `[4, 3, 2, 1]`.
2. **Complexité** :

- Déterminez la complexité de l'algorithme en fonction du nombre d'éléments.

Questions supplémentaires

- **Question 1** : Proposez une version de la fonction qui inverse la liste **in place** (c'est-à-dire sans créer de nouvelle liste).
- **Question 2** : Comparez la complexité des deux approches.
- **Question 3** : Expliquez pourquoi la boucle utilisée garantit la terminaison.

Exercice 2 : Vérification de palindrome

Objectif : Vérifier si une chaîne de caractères est un palindrome (se lit de la même façon de gauche à droite et de droite à gauche).

Consignes de base

1. **Fonction à réaliser** :
   Écrire une fonction `est_palindrome(chaine: str) -> bool` qui :
   - Ignore la casse et les espaces (vous pouvez utiliser des méthodes de chaîne pour transformer la chaîne en minuscules et supprimer les espaces).
   - Renvoie `True` si la chaîne est un palindrome, `False` sinon.
   - Exemple :
     `est_palindrome("Esope reste ici et se repose")` doit renvoyer `True`.
2. **Complexité** :
   - Analysez la complexité en fonction de la longueur de la chaîne.

Questions supplémentaires

- **Question 1** : Proposez une version récursive de la fonction.
- **Question 2** : Que se passe-t-il si la chaîne contient des signes de ponctuation ? Proposez une amélioration pour ne considérer que les lettres.
- **Question 3** : Expliquez pourquoi la terminaison de l'algorithme récursif est garantie.

Exercice 3 : Calcul du factoriel (itératif et récursif)

Objectif : Implémenter deux versions (itérative et récursive) d'une fonction calculant le factoriel d'un nombre entier.

Consignes de base

1. **Fonctions à réaliser** :
   - **Version itérative** : Écrire `factoriel_iteratif(n: int) -> int` qui calcule $n!$ par une boucle.
   - **Version récursive** : Écrire `factoriel_recursif(n: int) -> int` qui calcule $n!$ en s'appuyant sur la relation $n! = n \times (n-1)!$ avec $0! = 1$.
2. **Complexité** :

- Évaluer la complexité de chaque version en fonction de n.

Questions supplémentaires

- **Question 1** : Que se passe-t-il si `n` est négatif ? Proposez une gestion d'erreur dans ce cas.
- **Question 2** : Comparez les avantages et inconvénients des approches itérative et récursive.
- **Question 3** : Expliquez la terminaison de la fonction récursive.

Exercice 4 : Générateur de nombres premiers

Objectif : Créer une fonction qui renvoie la liste des nombres premiers inférieurs à un entier donné.

Consignes de base

1. **Fonction à réaliser** :
   Écrire une fonction `nombres_premiers(max_n: int) -> list` qui :
   - Pour chaque entier de 2 jusqu'à `max_n - 1`, vérifie s'il est premier (c'est-à-dire qu'il n'est divisible par aucun entier autre que 1 et lui-même).
   - Renvoie la liste des nombres premiers trouvés.
   - Exemple :
     `nombres_premiers(10)` doit renvoyer `[2, 3, 5, 7]`.
2. **Complexité** :
   - Analysez la complexité de l'algorithme dans sa forme naïve.

Questions supplémentaires

- **Question 1** : Expliquez comment l'algorithme du crible d'Ératosthène pourrait améliorer cette solution.
- **Question 2** : Que se passe-t-il si `max_n` est inférieur ou égal à 2 ?
- **Question 3** : Justifiez la terminaison de l'algorithme de vérification de primalité.

Exercice 5 : Extraction des chiffres d'un nombre

Objectif : Extraire les chiffres composant un nombre entier et les retourner dans une liste.

Consignes de base

1. **Fonction à réaliser** :
   Écrire une fonction `extraire_chiffres(n: int) -> list` qui :
   - Utilise la division entière et l'opérateur modulo pour extraire chaque chiffre.
   - Renvoie la liste des chiffres dans l'ordre d'apparition.
   - Exemple :
     `extraire_chiffres(12345)` doit renvoyer `[1, 2, 3, 4, 5]`.
2. **Complexité** :
   - Analysez la complexité de l'algorithme en fonction du nombre de chiffres.

Questions supplémentaires

- **Question 1** : Adaptez la fonction pour gérer correctement les nombres négatifs.
- **Question 2** : Proposez une version récursive de cette fonction.
- **Question 3** : Expliquez comment la boucle ou la récursion garantit la terminaison.

Exercice 6 : Fusion de deux dictionnaires avec somme des valeurs

Objectif : Fusionner deux dictionnaires en sommant les valeurs pour les clés communes.

Consignes de base

1. **Fonction à réaliser** :
   Écrire une fonction `fusionner_dictionnaires(d1: dict, d2: dict) -> dict` qui :
   - Parcourt les deux dictionnaires.
   - Pour chaque clé présente dans les deux dictionnaires, la valeur de la clé dans le dictionnaire final est la somme des deux valeurs.
   - Pour les clés présentes dans un seul dictionnaire, la valeur est directement reprise.
   - Exemple :
     `fusionner_dictionnaires({'a': 2, 'b': 3}, {'b': 4, 'c': 5})` doit renvoyer `{'a': 2, 'b': 7, 'c': 5}`.
2. **Complexité** :
   - Évaluez la complexité en fonction du nombre total d'éléments dans les dictionnaires.

Questions supplémentaires

- **Question 1** : Que faire si les valeurs ne sont pas numériques (par exemple, des chaînes) ?
- **Question 2** : Proposez des cas de tests (dictionnaires vides, dictionnaires avec plusieurs clés en commun, etc.).
- **Question 3** : Expliquez la terminaison de l'algorithme lors du parcours des dictionnaires.

Exercice 7 : Tri par insertion

Objectif : Implémenter l'algorithme de tri par insertion pour une liste d'entiers.

Consignes de base

1. **Fonction à réaliser** :
   Écrire une fonction `tri_insertion(liste: list) -> list` qui :
   - Parcourt la liste et insère chaque élément à sa place dans une sous-liste triée.
   - Renvoie la liste triée.
   - Exemple :
     `tri_insertion([4, 2, 5, 1])` doit renvoyer `[1, 2, 4, 5]`.
2. **Complexité** :

- Calculez la complexité moyenne (généralement en $O(n^2)$).

Questions supplémentaires

- **Question 1** : Comment se comporte l'algorithme sur une liste déjà triée ?
- **Question 2** : Proposez une amélioration ou expliquez pourquoi le tri par insertion est adapté aux petites listes.
- **Question 3** : Expliquez pourquoi la boucle d'insertion se termine toujours.

Exercice 8 : Comptage des mots dans une phrase

Objectif : Écrire une fonction qui compte le nombre de mots dans une phrase.

Consignes de base

1. **Fonction à réaliser** :
   Écrire une fonction `compter_mots(phrase: str) -> int` qui :
   - Considère qu'un mot est une séquence de caractères séparée par un ou plusieurs espaces.
   - Renvoie le nombre de mots dans la phrase.
   - Exemple :
     `compter_mots("Bonjour, comment ça va ?")` doit renvoyer `4` (en tenant compte d'une séparation par espaces).
2. **Complexité** :
   - Déterminez la complexité en fonction de la longueur de la phrase.

Questions supplémentaires

- **Question 1** : Comment traiteriez-vous le cas où la phrase contient plusieurs espaces consécutifs ou des signes de ponctuation collés aux mots ?
- **Question 2** : Proposez une variante qui renvoie aussi la liste des mots trouvés.
- **Question 3** : Expliquez en quoi la terminaison de l'algorithme est assurée lors du parcours de la chaîne.
