Thématique 1 - Tri et recherche

Sujet 1 : Tri par sélection

Énoncé : Écrire un algorithme qui trie une liste d'entiers par sélection.

Résolution : Le tri par sélection consiste à parcourir la liste, chercher le plus petit élément dans la partie non triée, puis l'échanger avec le premier élément non trié. On répète cette opération jusqu'à ce que toute la liste soit triée.

```
def tri_selection(tab):
    for i in range(len(tab)):
        mini = i
        for j in range(i+1, len(tab)):
            if tab[j] < tab[mini]:
                mini = j
        tab[i], tab[mini] = tab[mini], tab[i]
```

Sujet 2 : Trouver l'indice du maximum

Énoncé :Trouver l'indice du plus grand élément d'une liste d'entiers.

Résolution :On parcourt la liste en gardant l'indice du plus grand élément vu jusqu'à présent.

```
def indice_max(tab):
    max_i = 0
    for i in range(1, len(tab)):
        if tab[i] > tab[max_i]:
            max_i = i
    return max_i
```

Thématique 2 - Parcours de tableaux

Sujet 1 : Compter les occurrences

Énoncé :Compter combien de fois un élément donné apparaît dans une liste.

Résolution :On initialise un compteur à 0 et on l'incrémente à chaque fois qu'on rencontre l'élément.

```
def compter(tab, x):
    compteur = 0
    for valeur in tab:
        if valeur == x:
            compteur += 1
    return compteur
```

Sujet 2 : Trouver les indices d'un élément

Énoncé :Retourner la liste des indices où un élément donné apparaît dans une liste.

Résolution :On parcourt la liste, on ajoute chaque indice où l'élément apparaît dans une liste résultat.

```
def indices(tab, x):
    resultats = []
    for i in range(len(tab)):
        if tab[i] == x:
            resultats.append(i)
    return resultats
```

Thématique 3 - Programmation orientée objet (POO)

Sujet 1 : Classe Point

Énoncé :Créer une classe Point avec les attributs x et y, et une méthode pour calculer la distance à l'origine.

Résolution :Utiliser la formule de distance euclidienne.

```
import math

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_origine(self):
        return math.sqrt(self.x**2 + self.y**2)
```

Sujet 2 : Distance entre deux points

Énoncé :Ajouter à la classe Point une méthode pour calculer la distance entre deux points.

Résolution :Formule distance entre deux points (x1, y1) et (x2, y2) : racine carrée de la somme des carrés des différences.

```
def distance(self, autre):
    return math.sqrt((self.x - autre.x)**2 + (self.y - autre.y)**2)
```

Thématique 4 - Arbres binaires

Sujet 1 : Compter les feuilles

Énoncé :Compter le nombre de feuilles (nœuds sans enfants) dans un arbre binaire.

Résolution :Si un nœud n'a pas d'enfant gauche ni droit, c'est une feuille. Sinon, on compte les feuilles de ses sous-arbres.

```
def nombre_feuilles(arbre):
    if arbre is None:
        return 0
    if arbre.gauche is None and arbre.droit is None:
        return 1
    return nombre_feuilles(arbre.gauche) + nombre_feuilles(arbre.droit)
```

Sujet 2 : Calculer la hauteur

Énoncé :Calculer la hauteur (profondeur maximale) d'un arbre binaire.

Résolution :La hauteur d'un arbre est 1 + la hauteur maximale de ses sous-arbres.

```
def hauteur(arbre):
    if arbre is None:
        return 0
    return 1 + max(hauteur(arbre.gauche), hauteur(arbre.droit))
```

Thématique 5 - Parcours de graphes

Sujet 1 : Parcours en largeur (BFS) sans deque

Énoncé :Effectuer un parcours en largeur dans un graphe en utilisant une liste comme file.

Résolution :On utilise une liste pour stocker les sommets à visiter. On retire le premier élément (pop(0)) à chaque étape.

```
def parcours_largeur(graphe, depart):
    file = [depart]
    visites = []
    while file:
        sommet = file.pop(0)
        if sommet not in visites:
            visites.append(sommet)
            for voisin in graphe.get(sommet, []):
                if voisin not in visites and voisin not in file:
                    file.append(voisin)
    return visites
```

Sujet 2 : Parcours en profondeur (DFS) récursif

Énoncé :Effectuer un parcours en profondeur récursif dans un graphe.

Résolution :On visite un sommet, puis récursivement ses voisins non visités.

```
def parcours_profondeur(graphe, sommet, visites=None):
    if visites is None:
        visites = set()
    visites.add(sommet)
    for voisin in graphe.get(sommet, []):
        if voisin not in visites:
            parcours_profondeur(graphe, voisin, visites)
    return visites
```

Thématique 6 - SQL (bases de données)

Sujet 1 : Requête de sélection simple

Énoncé :Écrire une requête SQL qui affiche les noms et prénoms des élèves d'une classe.

Résolution :On utilise SELECT pour afficher des colonnes spécifiques.

```
SELECT nom, prenom FROM Eleves;
```

Sujet 2 : Sélection avec condition

Énoncé :Afficher les notes supérieures à 15 d'un élève identifié par son id.

Résolution :On utilise WHERE pour ajouter une condition sur l'id de l'élève et sur la note.

```
SELECT note FROM Notes WHERE eleve_id = 3 AND note > 15;
```

Thématique 7 - Tables de hachage (dictionnaires)

Sujet 1 : Fréquence des lettres

Énoncé :Écrire une fonction qui renvoie un dictionnaire contenant la fréquence de chaque lettre d'un mot.

Résolution :On initialise un dictionnaire, puis on parcourt le mot et on incrémente les occurrences.

```
def freq_lettres(mot):
    freq = {}
    for lettre in mot:
        if lettre in freq:
            freq[lettre] += 1
        else:
            freq[lettre] = 1
    return freq
```

Sujet 2 : Inverser un dictionnaire

Énoncé :Inverser un dictionnaire clé → valeur en valeur → liste de clés.

Résolution :On parcourt le dictionnaire et on ajoute chaque clé à la liste de la valeur correspondante.

```
def inverser(dico):
    inverse = {}
    for cle, valeur in dico.items():
        if valeur in inverse:
            inverse[valeur].append(cle)
        else:
            inverse[valeur] = [cle]
    return inverse
```

Thématique 8 - Chaînes de caractères

Sujet 1 : Palindrome

Énoncé :Vérifier si une chaîne de caractères est un palindrome (se lit dans les deux sens).

Résolution :On compare la chaîne avec son inverse.

```
def est_palindrome(chaine):
    return chaine == chaine[::-1]
```

Sujet 2 : Compter les mots

Énoncé :Compter combien de mots contient une chaîne de caractères.

Résolution :On utilise split() pour découper les mots.

```
def nombre_mots(phrase):
    return len(phrase.split())
```

Thématique 9 - Files et piles (structures linéaires)

Sujet 1 : Simuler une pile

Énoncé :Créer une pile avec append() pour empiler et pop() pour dépiler.

Résolution :Utiliser une liste comme pile (LIFO).

```
pile = []
pile.append(5)  # empiler
valeur = pile.pop()  # dépiler
```

Sujet 2 : Simuler une file sans deque

Énoncé :Créer une file avec append() pour ajouter et pop(0) pour retirer.

Résolution :Utiliser une liste comme file (FIFO).

```
file = []
file.append("A")  # enfile
valeur = file.pop(0)  # défile
```

Thématique 10 - Analyse de complexité

Sujet 1 : Complexité d'un double for

Énoncé :Analyser la complexité d'un algorithme contenant deux boucles imbriquées.

Résolution :Deux boucles for imbriquées = complexité $O(n^2)$.

```
for i in range(n):
    for j in range(n):
        faire_operation()
```

Sujet 2 : Complexité d'une recherche dichotomique

Énoncé :Quel est le nombre de comparaisons max pour chercher un élément dans une liste triée avec une recherche dichotomique ?

Résolution :À chaque étape, la taille de la liste est divisée par 2. Donc complexité $O(\log n)$.

```
def recherche_dicho(tab, x):
    debut, fin = 0, len(tab) - 1
    while debut <= fin:
        milieu = (debut + fin) // 2
        if tab[milieu] == x:
            return milieu
        elif tab[milieu] < x:
            debut = milieu + 1
        else:
            fin = milieu - 1
    return -1
```

1. Quelle est la différence entre une boucle for et une boucle while ?→ for est utilisée quand on connaît le nombre d'itérations, while quand on ne le connaît pas d'avance (condition).
2. Que doit contenir une fonction Python ?→ Un nom, des parenthèses avec paramètres (optionnels), le mot-clé def, une indentation, et souvent un return.
3. À quoi sert return ?→ À renvoyer une valeur en sortie de fonction.
4. Que fait range(3) ?→ Crée la séquence : 0, 1, 2.
5. Quelle est la différence entre append et extend ?→ append ajoute un élément, extend ajoute plusieurs (une liste).///
6. Quelle est la différence entre une liste et un dictionnaire ?→ Une liste est indexée par des entiers, un dictionnaire par des clés.
7. Comment accéder au 3e élément d'une liste L ?→ L[2].
8. Qu'est-ce qu'une pile ?→ Structure LIFO (dernier entré, premier sorti).
9. Qu'est-ce qu'une file ?→ Structure FIFO (premier entré, premier sorti).
10. Peut-on modifier un tuple ?→ Non, il est immuable.///
11. Qu'est-ce qu'une classe ?→ Un modèle pour créer des objets.
12. À quoi sert le mot-clé self ?→ À référencer l'instance en cours.
13. Que signifie __init__ ?→ C'est le constructeur, il initialise l'objet.
14. Quelle est la différence entre une méthode et une fonction ?→ Une méthode est liée à un objet (ex : mon_objet.methode()).///
15. Qu'est-ce qu'une fonction récursive ?→ Une fonction qui s'appelle elle-même.
16. Que doit contenir une fonction récursive ?→ Un cas de base et un appel récursif.
17. Pourquoi une récursion infinie est un problème ?→ Elle provoque un dépassement de pile (stack overflow).///
18. Qu'est-ce que le tri par sélection ?→ On cherche le plus petit élément et on l'échange avec le 1er, puis on répète.
19. Quelle est la complexité du tri par sélection ?→ $O(n^2)$
20. Quelle est la différence entre recherche linéaire et dichotomique ?→ Linéaire : élément par élément (O(n)),
Dichotomique : dans une liste triée, divise par 2 à chaque étape (O(log n)).///
21. Que fait la commande SELECT ?→ Elle sélectionne des données dans une table.
22. Que signifie WHERE ?→ C'est une condition pour filtrer les résultats.
23. Quelle est la différence entre DELETE et DROP ?→ DELETE enlève les données, DROP supprime la table entière.
24. Qu'est-ce qu'une clé primaire ?→ Un identifiant unique dans une table.///
25. Que signifie O(n), $O(n^2)$, O(log n) ?→ Ce sont des notations de complexité (nombre d'opérations en fonction de la taille des données).
26. Pourquoi le tri rapide (quicksort) est plus rapide que le tri par insertion ?→ Il divise les données (O(n log n)), l'insertion est quadratique ($O(n^2)$).///
27. Quelle est la différence entre un graphe et un arbre ?→ Un arbre est un graphe sans cycle.
28. Comment représenter un graphe ?→ Par matrice d'adjacence ou dictionnaire de listes.
29. C'est quoi un parcours en profondeur (DFS) ?→ On explore au maximum un chemin avant de revenir.
30. C'est quoi un parcours en largeur (BFS) ?→ On explore tous les voisins avant de descendre plus loin.