Algorithmique, initiation

Steeve Lefort

2024

Variables

Exercice 1: Création et Affichage

Créez un algorithme qui :

1. Déclare une variable pour l'âge (nombre entier)
2. Déclare une variable pour le prénom (chaîne de caractères)
3. Affecte des valeurs à ces variables
4. Affiche le contenu de ces variables

Exercice 2: Concaténation

Créez un algorithme qui :

1. Déclare et initialise une variable pour le prénom
2. Déclare et initialise une variable pour le nom
3. Crée une troisième variable qui combine le prénom et le nom
4. Affiche le résultat

Exercice 3: Calculs Simples

Créez un algorithme qui :

1. Déclare et initialise deux variables numériques
2. Effectue l'addition, la soustraction, la multiplication et la division de ces deux nombres
3. Affiche les résultats de chaque opération

Exercice 4: Calcul d'une Moyenne

Créez un algorithme qui :

1. Déclare trois variables pour stocker des notes
2. Permet à l'utilisateur de saisir les valeurs de ces variables
3. Calcule la moyenne de ces trois notes
4. Affiche le résultat

Exercice 5: Échange de Valeurs

Créez un algorithme qui :

1. Déclare deux variables A et B
2. Permet à l'utilisateur de saisir les valeurs de ces variables
3. Échange les valeurs de A et B
4. Affiche les valeurs avant et après l'échange

Illustration :

```
Avant :  A = 5  B = 10
Après :  A = 10 B = 5
```

Les tests conditionnels

Exercice 6: Le Gardien du Parc d'Attractions

Créez un algorithme pour un parc d'attractions qui :

1. Demande l'âge du visiteur
2. Si l'âge est inférieur à 12 ans, affiche "Bienvenue dans l'aire de jeux pour enfants !"
3. Si l'âge est entre 12 et 18 ans (inclus), affiche "Les montagnes russes vous attendent !"
4. Si l'âge est supérieur à 18 ans, affiche "Profitez de toutes nos attractions !"

Exercice 7: Le Conseiller en Cafés

Créez un algorithme pour une machine à café intelligente qui :

1. Demande à l'utilisateur s'il préfère un café fort ou léger
2. Demande s'il souhaite du lait
3. En fonction des réponses :
   - Si café fort sans lait : "Voici un expresso !"
   - Si café fort avec lait : "Voici un cappuccino !"
   - Si café léger sans lait : "Voici un café américain !"
   - Si café léger avec lait : "Voici un café latte !"

Les boucles

Exercice 8: Le Compteur de Pas

Créez un algorithme qui simule un podomètre :

1. Initialise le nombre de pas à 0
2. Demande à l'utilisateur combien de pas il a fait aujourd'hui
3. Ajoute ce nombre au total
4. Demande si l'utilisateur veut ajouter plus de pas
5. Si oui, répète les étapes 2-4
6. Si non, affiche le nombre total de pas

Exercice 9: La Table de Multiplication

Créez un algorithme qui :

1. Demande à l'utilisateur de choisir un nombre entre 1 et 10
2. Affiche la table de multiplication de ce nombre de 1 à 10
3. Pour chaque résultat pair, ajoute "Pair !" à côté du résultat

Les tableaux

Exercice 10: Calcul d'une Moyenne

Créez un algorithme qui :

1. Déclare un tableau destiné à stocker des notes
2. Permet à l'utilisateur de saisir autant de notes qu'il le souhaite
3. La saisie s'arrête si l'utilisateur saisi une valeur négative
4. Calcule la moyenne des notes
5. Affiche le résultat

Exercice 11: Le Gestionnaire d'Inventaire

Créez un algorithme pour gérer l'inventaire d'une petite boutique :

1. Initialisez un tableau de 5 produits avec leurs quantités
2. Affichez l'inventaire initial
3. Demandez à l'utilisateur quel produit il souhaite acheter (par index)
4. Demandez la quantité souhaitée
5. Mettez à jour l'inventaire et affichez-le à nouveau

Exercice 12: L'Analyseur de Températures

Créez un algorithme qui analyse les températures d'une semaine :

1. Initialisez un tableau de 7 températures (une pour chaque jour)
2. Trouvez et affichez la température la plus élevée et la plus basse
3. Calculez et affichez la température moyenne de la semaine

Exercice 13: Transposition de matrice

Objectif

Écrire un algorithme qui transpose une matrice 3x3.

Définition

La transposition d'une matrice consiste à échanger ses lignes et ses colonnes. Dans une matrice transposée, le premier élément de chaque ligne devient le premier élément de chaque colonne, le deuxième élément de chaque ligne devient le deuxième élément de chaque colonne, et ainsi de suite.

Exemple

```
Matrice originale :        Matrice transposée :
| 1 2 3 |                  | 1 4 7 |
| 4 5 6 |                  | 2 5 8 |
| 7 8 9 |                  | 3 6 9 |
```

Instructions

1. Créez une matrice 3x3 avec des valeurs de votre choix.
2. Affichez la matrice originale.
3. Créez une nouvelle matrice pour stocker le résultat de la transposition.
4. Parcourez la matrice originale et remplissez la nouvelle matrice en échangeant les indices des lignes et des colonnes.
5. Affichez la matrice transposée.

Les procédures et fonctions

Exercice 14: Le Convertisseur de Devises

Créez un algorithme avec une fonction qui convertit des euros en dollars :

1. Créez une fonction "euroToDollar" qui prend un montant en euros et renvoie l'équivalent en dollars
2. Dans le programme principal, demandez à l'utilisateur un montant en euros
3. Utilisez la fonction pour convertir et afficher le résultat en dollars

Exercice 15: Le Calculateur d'IMC

Créez un algorithme avec une procédure qui calcule l'Indice de Masse Corporelle (IMC) :

1. Créez une procédure "calculIMC" qui prend le poids et la taille et affiche l'IMC
2. Dans le programme principal, demandez à l'utilisateur son poids et sa taille
3. Utilisez la procédure pour calculer et afficher l'IMC

IMC

La formule pour calculer l'Indice de Masse Corporelle (IMC) est:

$$\texttt{IMC = poids / (taille}^2\texttt{)}$$

La récursivité

Exercice 16: La Suite de Fibonacci (partie 1)

Cet exercice n'est pas réalisable avec Algobox. Nous allons l'implémenter en Python.

La suite de Fibonacci est définie par la relation de récurrence suivante :

```
F(n) = F(n-1) + F(n-2)
```

avec les conditions initiales :
$F(0) = 0$ $F(1) = 1$
Les premiers termes de la suite de Fibonacci sont :

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
```

```
F(0) = 0 (condition initiale)
F(1) = 1 (condition initiale)
F(2) = F(1) + F(0) = 1 + 0 = 1
F(3) = F(2) + F(1) = 1 + 1 = 2
F(4) = F(3) + F(2) = 2 + 1 = 3
F(5) = F(4) + F(3) = 3 + 2 = 5
```

Et ainsi de suite...

Exercice 16: La Suite de Fibonacci (partie 2)

Créez un algorithme récursif qui calcule le n-ième terme de la suite de Fibonacci :

1. Créez une fonction récursive "fibonacci" qui prend un nombre n en entrée
2. La fonction doit renvoyer le n-ième terme de la suite de Fibonacci
3. Dans le programme principal, demandez à l'utilisateur un nombre n
4. Utilisez la fonction pour calculer et afficher le résultat

Exercice 17: Le Compteur à Rebours Récursif

Créez un algorithme récursif qui effectue un compte à rebours :

1. Créez une procédure récursive "compteARebours" qui prend un nombre n en entrée
2. La procédure doit afficher les nombres de n à 1, puis "Décollage !"
3. Dans le programme principal, demandez à l'utilisateur un nombre de départ
4. Utilisez la procédure pour effectuer le compte à rebours

Piles et files

Exercice 18: Simulation d'une pile de livres

Objectif : Implémenter les opérations de base d'une pile (empiler et dépiler) en utilisant un tableau.
Instructions :

1. Créez un tableau pour représenter une pile de livres.
2. Implémentez une fonction pour ajouter un livre au sommet de la pile (empiler).
3. Implémentez une fonction pour retirer le livre du sommet de la pile (dépiler).
4. Affichez l'état de la pile après chaque opération.

Exemple d'utilisation :

- Empilez les livres : "Harry Potter", "Le Seigneur des Anneaux", "1984"
- Dépilez un livre
- Empilez "Dune"
- Affichez le contenu final de la pile

Exercice 19: Vérification des parenthèses

Objectif : Utiliser une pile pour vérifier si une expression mathématique a des parenthèses bien équilibrées.

Instructions:

1. Créez une fonction qui prend une chaîne de caractères représentant une expression mathématique.
2. Parcourez la chaîne caractère par caractère.
3. Si vous rencontrez une parenthèse ouvrante '(', empilez-la.:
4. Si vous rencontrez une parenthèse fermante ')':
   - Si la pile est vide, l'expression est mal équilibrée (parenthèse fermante en trop).
   - Sinon, dépilez une parenthèse ouvrante.
5. À la fin du parcours :
   - Si la pile est vide, l'expression est bien équilibrée.
   - Si la pile n'est pas vide, l'expression est mal équilibrée (parenthèses ouvrantes en trop).

Exemple d'utilisation :

- Testez avec l'expression : "((a+b)*(c-d))" (bien équilibrée)
- Testez avec l'expression : "((a+b)*(c-d)" (parenthèse ouvrante en trop)
- Testez avec l'expression : "((a+b)*(c-d)))" (parenthèse fermante en trop)

Exercice 20: Simulation d'une file d'attente

Objectif : Implémenter les opérations de base d'une file (enfiler et défiler) en utilisant un tableau.
Instructions :

1. Créez un tableau pour représenter une file d'attente de clients dans un magasin.
2. Implémentez une fonction pour ajouter un client à la fin de la file (enfiler).
3. Implémentez une fonction pour retirer le client en tête de file (défiler).
4. Affichez l'état de la file après chaque opération.

Exemple d'utilisation :

- Enfilez les clients : "Alice", "Bob", "Charlie"
- Défilez un client (le premier arrivé est le premier servi)
- Enfilez "David"
- Affichez le contenu final de la file

Exercice 21: Inversion d'une chaîne de caractères

Objectif : Utiliser une pile pour inverser une chaîne de caractères.
Instructions :

1. Créez une fonction qui prend une chaîne de caractères en entrée.
2. Parcourez la chaîne et empilez chaque caractère.
3. Créez une nouvelle chaîne en dépilant les caractères un par un.
4. Retournez la nouvelle chaîne (qui sera l'inverse de l'originale).

Exemple d'utilisation :

- Testez avec la chaîne : "algorithmique"
- Affichez la chaîne inversée

POO

Exercice 22: Création d'une classe Personnage

Créez une classe `Personnage` pour un jeu vidéo avec les caractéristiques suivantes :

- Propriétés : `nom`, `points_de_vie`, `force`
- Constructeur : initialise les propriétés lors de la création d'un personnage
- Méthodes :
  - `afficher_infos()` : affiche les informations du personnage
  - `attaquer(cible)` : réduit les points de vie de la cible en fonction de la force du personnage

Créez ensuite deux instances de personnages et faites-les interagir.

Complexité algorithmique

Exercice 23: Évaluation de la complexité

Évaluez la complexité de l'algorithme suivant :

```
VARIABLES
    tableau EST_DU_TYPE TABLEAU
    taille EST_DU_TYPE NOMBRE
    premier_element EST_DU_TYPE NOMBRE

DEBUT_ALGORITHME
    taille PREND_LA_VALEUR 1000
    POUR i ALLANT_DE 1 A taille
        tableau[i] PREND_LA_VALEUR i
    FIN_POUR

    premier_element PREND_LA_VALEUR tableau[1]
    AFFICHER premier_element
FIN_ALGORITHME
```

Exercice 24: Évaluation de la complexité

Évaluez la complexité de l'algorithme suivant :

```
VARIABLES
    tableau EST_DU_TYPE TABLEAU
    taille EST_DU_TYPE NOMBRE
    somme EST_DU_TYPE NOMBRE

DEBUT_ALGORITHME
    taille PREND_LA_VALEUR 1000
    somme PREND_LA_VALEUR 0

    POUR i ALLANT_DE 1 A taille
        tableau[i] PREND_LA_VALEUR i
        somme PREND_LA_VALEUR somme + tableau[i]
    FIN_POUR

    AFFICHER somme
FIN_ALGORITHME
```

Exercice 25: Évaluation de la complexité

Évaluez la complexité de l'algorithme suivant :

```
VARIABLES
    tableau EST_DU_TYPE TABLEAU
    taille EST_DU_TYPE NOMBRE
    somme EST_DU_TYPE NOMBRE
    premier_element EST_DU_TYPE NOMBRE

DEBUT_ALGORITHME
    taille PREND_LA_VALEUR 1000
    somme PREND_LA_VALEUR 0

    POUR i ALLANT_DE 1 A taille
        tableau[i] PREND_LA_VALEUR i
        somme PREND_LA_VALEUR somme + tableau[i]
    FIN_POUR

    premier_element PREND_LA_VALEUR tableau[1]
    AFFICHER somme
    AFFICHER premier_element
FIN_ALGORITHME
```

Exercice 26: Évaluation de la complexité

Évaluez la complexité de l'algorithme suivant :

```
VARIABLES
    tableau EST_DU_TYPE TABLEAU
    taille EST_DU_TYPE NOMBRE
    somme EST_DU_TYPE NOMBRE

DEBUT_ALGORITHME
    taille PREND_LA_VALEUR 1000
    somme PREND_LA_VALEUR 0

    POUR i ALLANT_DE 1 A taille
        tableau[i] PREND_LA_VALEUR i
    FIN_POUR

    POUR i ALLANT_DE 1 A taille
        POUR j ALLANT_DE 1 A taille
            somme PREND_LA_VALEUR somme + tableau[i] * tableau[j]
        FIN_POUR
    FIN_POUR

    AFFICHER somme
FIN_ALGORITHME
```

Les tris

Exercice 27: Implémentation du Tri à Bulles

Objectif : Implémenter l'algorithme du tri à bulles pour trier un tableau de nombres.
Instructions :

1. Créez une fonction "triBulles" qui prend un tableau en entrée.
2. Implémentez l'algorithme du tri à bulles comme décrit dans les slides.
3. La fonction doit modifier le tableau d'origine pour le trier par ordre croissant.
4. Affichez le tableau avant et après le tri.

Exemple d'utilisation :

- Testez avec le tableau : [64, 34, 25, 12, 22, 11, 90]

Exercice 28: Tri par Sélection

Objectif : Implémenter l'algorithme du tri par sélection.
Instructions :

1. Créez une fonction "triSelection" qui prend un tableau en entrée.
2. Implémentez l'algorithme du tri par sélection comme décrit dans les slides.
3. La fonction doit modifier le tableau d'origine pour le trier par ordre croissant.
4. Affichez le tableau avant et après le tri.

Exemple d'utilisation :

- Testez avec le tableau : [64, 25, 12, 22, 11]

Exercice 29: Tri par Insertion

Objectif : Implémenter l'algorithme du tri par insertion.
Instructions :

1. Créez une fonction "triInsertion" qui prend un tableau en entrée.
2. Implémentez l'algorithme du tri par insertion comme décrit dans les slides.
3. La fonction doit modifier le tableau d'origine pour le trier par ordre croissant.
4. Affichez le tableau avant et après le tri.

Exemple d'utilisation :

- Testez avec le tableau : [12, 11, 13, 5, 6]

Exercice 30: Implémentation du Tri Fusion

Objectif : Implémenter l'algorithme du tri fusion (merge sort).
Instructions :

1. Créez une fonction "fusion" qui fusionne deux sous-tableaux triés.
2. Créez une fonction récursive "triFusion" qui implémente l'algorithme du tri fusion.
3. La fonction doit retourner un nouveau tableau trié.
4. Affichez le tableau avant et après le tri.

Exemple d'utilisation :

- Testez avec le tableau : [38, 27, 43, 3, 9, 82, 10]

Exercice 31: Implémentation du Tri Rapide (Quicksort)

Objectif : Implémenter l'algorithme du tri rapide (quicksort).
Instructions :

1. Créez une fonction "partition" qui choisit un pivot et partitionne le tableau.
2. Créez une fonction récursive "triRapide" qui implémente l'algorithme du tri rapide.
3. La fonction doit modifier le tableau d'origine pour le trier par ordre croissant.
4. Affichez le tableau avant et après le tri.

Exemple d'utilisation :

- Testez avec le tableau : [10, 7, 8, 9, 1, 5]

Exercice 32: Implémentation du Tri par Tas (Heap Sort)

Objectif : Implémenter l'algorithme du tri par tas (heap sort).
Instructions :

1. Créez une fonction "entasser" qui maintient la propriété de tas pour un sous-arbre enraciné à l'indice i.
2. Créez une fonction "construireMaxTas" qui construit un tas max à partir du tableau.
3. Créez une fonction "triTas" qui implémente l'algorithme du tri par tas.
4. La fonction doit modifier le tableau d'origine pour le trier par ordre croissant.
5. Affichez le tableau avant et après le tri.

Exemple d'utilisation :

- Testez avec le tableau : [12, 11, 13, 5, 6, 7]

Exercice 33: Comparaison du nombre de swaps

Objectif : Comparer l'efficacité des algorithmes de tri en termes de nombre d'échanges (swaps) effectués.
Implémentez les algorithmes de tri à bulles, tri par sélection et tri par insertion. Pour chaque algorithme :

1. Ajoutez un compteur pour suivre le nombre d'échanges effectués.
2. Triez le tableau [5, 2, 8, 12, 1, 6] par ordre croissant.
3. Affichez le nombre total d'échanges réalisés.

Comparez les résultats et discutez de l'efficacité relative de chaque algorithme en termes de nombre d'échanges.

Exercice 34: Comparaison des Algorithmes de Tri

Instructions

- Utilisez ce code pour mesurer le temps d'exécution :

```
import time

def mesurer_temps(fonction_tri, tableau):
    debut = time.time()
    fonction_tri(tableau.copy())
    fin = time.time()
    return fin - debut

# Exemple d'utilisation
tableau = [5, 2, 8, 12, 1, 6]
temps = mesurer_temps(votre_fonction_de_tri, tableau)
print(f"Temps d'exécution : {temps:.6f} secondes")
```

- Mesurez le temps pour chaque algorithme (tri à bulles, sélection, insertion).
- Comparez les résultats : Quel algorithme est le plus rapide ? Le plus lent ?
- Testez avec des tableaux de différentes tailles (100, 1000, 5000 éléments).
- Pourquoi certains algorithmes sont-ils plus rapides que d'autres ?
- Comment les performances changent-elles avec la taille du tableau ?

Les arbres

Exercice 35: Implémentation d'un Arbre Binaire

Objectif : Implémenter une structure d'arbre binaire et des fonctions pour l'insertion et l'affichage des nœuds.
Instructions :

1. Créez une structure Noeud avec les attributs valeur, gauche et droite.
2. Implémentez une fonction pour insérer un nouveau nœud dans l'arbre.
3. Implémentez une fonction pour afficher l'arbre en utilisant un parcours en profondeur (récursif).
4. Dans le programme principal, créez un arbre binaire et insérez-y les valeurs suivantes : 5, 3, 7, 1, 9, 4, 6
5. Affichez l'arbre résultant.

Exemple d'utilisation :

- Créez l'arbre binaire avec les valeurs données.
- Affichez l'arbre pour vérifier la structure.

Exercice 36: Parcours en Largeur d'un Arbre Binaire

Objectif : Implémenter un parcours en largeur (BFS) d'un arbre binaire.
Instructions :

1. Utilisez la structure Noeud de l'exercice précédent.
2. Implémentez une fonction `parcours_largeur` qui effectue un parcours en largeur de l'arbre.
3. Utilisez une file pour stocker les nœuds à visiter.
4. Dans le programme principal, créez le même arbre que dans l'exercice précédent.
5. Effectuez un parcours en largeur de l'arbre et affichez les valeurs des nœuds dans l'ordre de visite.

Exemple d'utilisation :

- Créez l'arbre binaire avec les valeurs : 5, 3, 7, 1, 9, 4, 6
- Effectuez un parcours en largeur et affichez le résultat.

Exercice 37: Parcours en Profondeur Récursif d'un Arbre Binaire

Objectif : Implémenter les trois types de parcours en profondeur récursifs d'un arbre binaire.
Instructions :

1. Utilisez la structure Noeud des exercices précédents.
2. Implémentez trois fonctions récursives pour les parcours en profondeur : a. `parcours_preordre` (racine, gauche, droite) b. `parcours_inordre` (gauche, racine, droite) c. `parcours_postordre` (gauche, droite, racine)
3. Dans le programme principal, créez le même arbre que dans les exercices précédents.
4. Effectuez les trois types de parcours et affichez les résultats.

Exemple d'utilisation :

- Créez l'arbre binaire avec les valeurs : 5, 3, 7, 1, 9, 4, 6
- Effectuez les trois types de parcours et affichez les résultats.

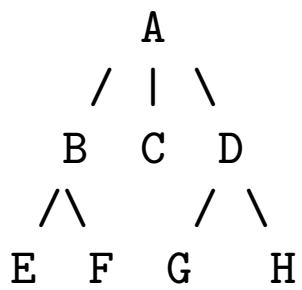
Exercice 38: Backtracking sur un Arbre N-aire

Objectif : Implémenter un algorithme de backtracking pour trouver tous les chemins de la racine aux feuilles dans un arbre n-aire.
Instructions :

1. Créez une structure NoeudNAire avec les attributs valeur et enfants (liste de nœuds).
2. Implémentez une fonction récursive $\text{trouver}_{\text{chemins}}$ qui utilise le backtracking pour trouver tous les chemins de la racine aux feuilles.
3. La fonction doit accumuler le chemin actuel et l'ajouter à la liste des chemins trouvés lorsqu'une feuille est atteinte.
4. Dans le programme principal, créez un arbre n-aire simple et trouvez tous les chemins.

Exemple d'utilisation :

- Créez un arbre n-aire avec la structure suivante :



- Trouvez tous les chemins de la racine aux feuilles.

Les graphes

Exercice 39: Implémentation d'un Graphe

Objectif : Implémenter une structure de graphe à l'aide d'une liste d'adjacence.
Instructions :

1. Créez une classe Graphe avec une méthode pour ajouter des sommets et des arêtes.
2. Implémentez une méthode pour afficher le graphe.
3. Dans le programme principal, créez un graphe représentant le réseau routier des villes suivantes : Paris, Lille, Rennes, Tours, Strasbourg, Bourges.
4. Ajoutez les connexions entre les villes selon la carte fournie dans les slides.
5. Affichez le graphe résultant.

Exemple d'utilisation :

- Créez le graphe du réseau routier.
- Affichez la liste d'adjacence pour vérifier la structure.

Exercice 40: Parcours en Largeur (BFS)

Objectif : Implémenter l'algorithme de parcours en largeur (BFS) sur le graphe du réseau routier.
Instructions :

1. Utilisez la structure de Graphe de l'exercice précédent.
2. Implémentez une fonction bfs qui prend en paramètres le graphe et un sommet de départ.
3. La fonction doit afficher les sommets dans l'ordre de leur découverte.
4. Dans le programme principal, créez le même graphe que dans l'exercice précédent.
5. Effectuez un parcours en largeur à partir de Paris et affichez le résultat.

Exemple d'utilisation :

- Créez le graphe du réseau routier.
- Effectuez un parcours en largeur à partir de Paris et affichez le résultat.

Exercice 41: Algorithme de Kruskal

Objectif : Implémenter l'algorithme de Kruskal pour trouver l'arbre couvrant de poids minimum du réseau routier.
Instructions :

1. Implémentez une fonction kruskal qui prend en paramètre le graphe.
2. La fonction doit retourner les arêtes de l'arbre couvrant de poids minimum.
3. Dans le programme principal, créez le même graphe que dans les exercices précédents.
4. Appliquez l'algorithme de Kruskal et affichez les arêtes de l'arbre couvrant de poids minimum.

Exemple d'utilisation :

- Créez le graphe du réseau routier.
- Appliquez l'algorithme de Kruskal et affichez le résultat.

Exercice 42: Algorithme de Floyd-Warshall

Objectif : Implémenter l'algorithme de Floyd-Warshall pour trouver les plus courts chemins entre toutes les paires de sommets du réseau routier.
Instructions :

1. Implémentez une fonction `floyd_warshall` qui prend en paramètre le graphe.
2. La fonction doit retourner une matrice des distances les plus courtes entre chaque paire de sommets.
3. Dans le programme principal, créez le même graphe que dans les exercices précédents.
4. Appliquez l'algorithme de Floyd-Warshall et affichez la matrice des distances résultante.

Exemple d'utilisation :

- Créez le graphe du réseau routier.
- Appliquez l'algorithme de Floyd-Warshall et affichez la matrice des distances.

TP complémentaires

Introduction

Objectifs des projets

- Implémenter trois jeux classiques en Python
- Appliquer les concepts fondamentaux de l'algorithmique
- Créer des programmes interactifs en mode texte

Contraintes communes

Pour tous les projets

- Python standard uniquement
- Mode texte (console)
- Gestion des erreurs de saisie
- Code commenté et structuré

Projet 1 : Le Jeu du Pendu

Règles et spécifications

- Le programme permet la saisie d'un mot par un joueur
- Les autres joueurs devinent lettre par lettre
- Nombre limité d'essais

Projet 2 : Devine le Nombre

Règles et spécifications

- Programme choisit un nombre (1-100)
- Joueur essaie de deviner
- Indices : trop haut/bas
- Le nombre de tentatives est compté et affiché

Projet 3 : Morpion (Tic-Tac-Toe)

Règles et spécifications

- Jeu à deux joueurs (X et O)
- Chaque joueur saisi à son tour les coordonnées
- Grille 3x3
- Victoire : aligner 3 symboles

Fin des exercices

Félicitations !