Première partie, questions à choix unique

Pour chaque question marquer la case correspondante à la réponse correcte sans faire de ratures.

Question 1 Quelle est la sortie du programme Python suivant ?

```
l1 = [5, 2, 8, 3, 1, 7]
l2 = [2, 6, 3, 1, 7, 4]

path_map = {}
final_destinations = {}

for i in range(len(l1)):
    start = l1[i]
    end = l2[i]
    path_map[start] = end

for key in path_map:
    destination = path_map[key]
    while destination in path_map:
        destination = path_map[destination]
    final_destinations[key] = destination

print(final_destinations[3])
```

- ☐ 8
- ☐ 4
- ☐ 1
- ☐ 6

Question 2 Quel est le principal défaut de la fonction suivante:

```
def f(n):
    m = 1
    if n==1:
        return m
    for i in range(1,n):
        m = m + f(i)
    return m
```

- ☐ Elle ne s'arrête jamais.
- ☐ Elle est récursive.
- ☐ Elle a une complexité exponentielle.
- ☐ Elle sort tout le temps la même chose.

Question 3 Un aventurier explore une grotte remplie de coffres pour trouver un trésor caché. Parmi **12 coffres**, seuls **3 contiennent un trésor** (les autres sont vides, sauf un qui contient une bombe déclenchant un échec immédiat).
L'aventurier suit la stratégie suivante :

- Il examine les coffres dans un ordre aléatoire.
- Parmi les coffres contenant des pierres précieuses, l'un contient **3 kg**, un autre **4 kg** et le dernier **5 kg** et l'aventurier doit **cumuler au moins 8 kilogrammes de pierres précieuses** en ouvrant suffisamment de coffres avec des pierres précieuses.
- L'aventurier a **6 tentatives maximum** avant que la grotte ne s'effondre.

On souhaite simuler cette situation par programme. **Complétez le code suivant** en remplaçant le bloc `# A COMPLETER` par l'option correcte.

```python
import random

def simulate_one_trial():
    chests = [3, 4, 5] + [0] * 8 + [-1]  # 3 coffres avec pierres precieuses
                                         # et 1 bombe
    random.shuffle(chests)

    gem_count = 0
    for attempt in range(6):
        # A COMPLETER

    return False
```

Quelle option est correcte ?

☐
```
gem_count += 1
if chests[attempt] == -1:
    return False
if gem_count >= 8:
    return True
```

☐
```
gem_count += chests[attempt]
if chests[attempt] < 0:
    return False
if gem_count >= 8:
    return True
```

☐
```
gem_count += chests[attempt]
if gem_count >= 8:
    return True
if chests[attempt] == -1:
    return False
```

☐
```
gem_count += chests[attempt]
if attempt == 5:
    return False
if gem_count >= 8:
    return True
```

Question 4 On souhaite déterminer le chemin de coût minimal pour atteindre la case en bas à droite d'un tableau à deux dimensions contenant des entiers positifs. On ne peut se déplacer que vers le bas ou vers la droite. Voici deux implémentations : une approche gloutonne et une approche dynamique.

Code 1 : Algorithme Glouton

```python
def chemin_minimal_glouton(tab: list[list[int]]) -> int:
    n, m = len(tab), len(tab[0])
    i, j = 0, 0
    cout = tab[0][0]

    while i < n - 1 or j < m - 1:
        if i == n - 1:  # Derniere ligne, on ne peut aller qu'a droite
            j += 1
        elif j == m - 1:  # Derniere colonne, on ne peut aller qu'en bas
            i += 1
        elif tab[i + 1][j] < tab[i][j + 1]:
            i += 1
        else:
            j += 1
        cout += tab[i][j]

    return cout
```

Code 2 : Programmation Dynamique

```python
def chemin_minimal_dynamique(tab: list[list[int]]) -> int:
    n, m = len(tab), len(tab[0])
    dp = [[0] * m for _ in range(n)]  # Initialise tous les elements a zero

    dp[0][0] = tab[0][0]

    for i in range(1, n):
        dp[i][0] = dp[i-1][0] + tab[i][0]
    for j in range(1, m):
        dp[0][j] = dp[0][j-1] + tab[0][j]

    for i in range(1, n):
        for j in range(1, m):
            dp[i][j] = tab[i][j] + min(dp[i-1][j], dp[i][j-1])

    return dp[-1][-1]
```

a) Laquelle des entrées suivantes produit une réponse **non optimale** avec l'algorithme glouton mais **optimale** avec l'algorithme dynamique ?

- ☐ `[[1, 3, 1], [1, 5, 1], [4, 2, 1]]`
- ☐ `[[1, 2, 3], [4, 8, 2], [1, 5, 3]]`
- ☐ `[[ 1, 10,  1], [ 1,  1, 100], [10,  1,  1]]`
- ☐ `[[1,  1, 1], [1, 100, 1], [1,  1, 1]]`

b) Quelle est respectivement la complexité temporelle de chaque algorithme, en fonction de $n =$ `len(tab)` et $m =$ `len(tab[0])` ?

- ☐ Glouton: $\Theta(nm)$, Dynamique: $\Theta(nm \log(nm))$
- ☐ Glouton: $\Theta(n+m)$, Dynamique: $\Theta(nm)$
- ☐ Glouton: $\Theta(n+m)$, Dynamique: $\Theta(n+m)$
- ☐ Glouton: $\Theta(nm)$, Dynamique: $\Theta(n+m)$

Question 5

Quel sera le contenu de la variable a à la fin de l'exécution du programme ci-après ?

```
a = "10"
b = "6"
b = a
temp = a + b
a = temp
```

- [ ] 20
- [ ] 106
- [ ] 1010
- [ ] 12

Question 6

On considère l'extrait de code ci-dessous. Comment compléter ce programme pour que le contenu de la variable 'resultat' soit 'oaaio' à la fin de son exécution ?

```
s = "programmation"
resultat = ""
for ...: # Completer ici
    if loop_var in ("a", "e", "i", "o", "u", "y"):
        resultat += loop_var
```

- [ ] `loop_var in range(s)`
- [ ] `loop_var in s`
- [ ] `loop_var in len(s)`
- [ ] `loop_var in range(len(s))`

Question 7

Quelle est la sortie du programme suivant après son exécution ?

```
mot = "Iteratif"
output = ""
for i in range(0, len(mot), 2):
    output += mot[i]
print(output)
```

- [ ] Iteratif
- [ ] Ieai
- [ ] trtf
- [ ] Iter

Question 8

On considère l'extrait de code ci-dessous. Quelle expression sera toujours évaluée à True, en partant du principe que a et b sont deux variables de type int strictement positives?

```
q = a // b
r = a % b
```

- [ ] q * b + r == a
- [ ] q * b == a + r
- [ ] q // r * b == a
- [ ] q * b == a

Question 9

Qu'affiche ce code ?

```python
def is_adult(age: int) -> bool:
    return age >= 18

def is_retired(age: int) -> bool:
    return age >= 65

my_age: int = 21
if is_adult(my_age):
    print("A")
elif not is_retired(my_age):
    print("B")
else:
    print("C")
```

- [ ] A
  B
- [ ] B
- [ ] C
- [ ] A

Question 10

Laquelle de ces affirmations à propos du code ci-dessous est correcte ?

```python
a = input("Entrez un nombre: ")
b = input("Entrez un autre nombre: ")
print(a + b)
```

- [ ] Ce code ne fonctionne pas ainsi
- [ ] Ce code effectue toujours une addition
- [ ] Ce code effectue soit une addition soit une concaténation suivant ce que l'on tape
- [ ] Ce code effectue toujours une concaténation de strings

Deuxième partie, questions de type ouvert

Répondre dans l'espace dédié. Votre réponse doit être soigneusement justifiée, toutes les étapes de votre raisonnement doivent figurer dans votre réponse. Laisser libres les cases à cocher : elles sont réservées au correcteur.

Question 11: *Cette question est notée sur 4 points.*

☐0 ☐1 ☐2 ☐3 ☐4

On souhaite implémenter une fonction **récursive** permettant de compter les occurrences d'un caractère donné dans une chaîne de caractères.
a) (1 point) Implémentez une fonction **récursive** `compte_caractere()` qui prend en argument une chaîne de caractères `s` et un caractère `c`, et retourne le nombre de fois que `c` apparaît dans `s`.

```
def compte_caractere(s: str, c: str) -> int:
```

b) (2 points) En utilisant la fonction précédente, implémentez la fonction suivante qui compte le nombre total d'occurrences d'un caractère donné dans une liste de chaînes de caractères.

```
def compte_total(chaine_list: list[str], c: str) -> int:
```

c) (1 point) Écrivez du code qui utilise la fonction `compte_total()` pour afficher le nombre total de fois que la lettre `'a'` apparaît dans la liste suivante :

```
texte = ["apple", "banana", "avocado", "grape", "orange"]
```

Vous pouvez utiliser cette page au cas où vous n'auriez pas assez de place pour répondre à une question.

Pour faciliter la correction, **merci de bien indiquer** *à la question donnée qu'il faut venir lire la suite ici !*

Question 12: *Cette question est notée sur 5 points.*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5

Dans cet exercice, nous souhaitons implémenter une version simplifiée et incomplète du jeu Mastermind. Le

jeu consiste à deviner une séquence secrète de 4 chiffres (entre 0 et 9) générée aléatoirement par l'ordinateur. À chaque tentative, le joueur propose une séquence de 4 chiffres et reçoit en retour :

- Le nombre de chiffres bien placés (c'est-à-dire ayant la bonne valeur et à la bonne position) ;
- Le nombre de chiffres corrects mais mal placés (ayant la bonne valeur mais à la mauvaise position).

Par exemple, supposons que la séquence secrète soit **1234** :

- Si le joueur propose **1256**, la réponse est : 2 bien placés (1 et 2) et 0 mal placés.
- Si le joueur propose **4321**, la réponse est : 0 bien placé et 4 mal placés.
- Si le joueur propose **1234**, la réponse est : 4 bien placés (le joueur a gagné !).

a) (2 points) Écrivez une fonction `generate_secret_code() -> str` qui génère et retourne une séquence secrète de 4 chiffres aléatoires.

```
def generate_secret_code() -> str:
```

b) (2 points) Écrivez une fonction `evaluate_guess(secret: str, guess: str) -> tuple[int, int]` qui prend en paramètre la séquence secrète `secret` et une proposition `guess`, et qui retourne une liste contenant :

- Le nombre de chiffres bien placés à l'indice 0
- Le nombre de chiffres corrects mais mal placés à l'indice 1

```
def evaluate_guess(secret: str, guess: str) -> list[int]:
```

c) (1 point) Analysez la complexité de votre fonction `evaluate_guess`. Justifiez votre réponse.

Question 12: *Cette question est notée sur 4 points.*

☐0 ☐1 ☐2 ☐3 ☐4

On construit un petit système de gestion de notes d'élèves. Un élève a une liste de notes pour chaque matière.

a) (1 point) La classe suivante stocke le nom d'un cours et une liste de notes associées. Ajoutez-y (de manière correctement indentée) une méthode `avg_grade()` qui retourne la moyenne des notes. Si la liste de notes est vide, la méthode doit retourner `-1`.

```
@dataclass
class GradeRecord:
      course_name: str
      grades: list[float]
```

Pour le reste de la question, considérez que nous avons en plus le code suivant:

```
@dataclass
class Student:
    id: str
    first_name: str
    last_name: str
    grade_records: list[GradeRecord]

student1 = Student(id="S001", first_name="Jane", last_name="Doe", grade_records=[
    GradeRecord("Math",    grades=[6.0, 6.0, 6.0]),
    GradeRecord("English", grades=[4.5, 5.0, 6.0]),
    GradeRecord("History", grades=[6.0, 6.0, 6.0]),
    GradeRecord("Physics", grades=[5.0, 5.5, 5.0]),
])

student2 = Student(id="S002", first_name="John", last_name="Smith", grade_records=[
    GradeRecord("Math",    grades=[5.5, 6.0, 4.5]),
    GradeRecord("English", grades=[5.5, 5.5, 5.5]),
    GradeRecord("History", grades=[4.5, 5.0, 5.5]),
    GradeRecord("Physics", grades=[5.0, 5.5, 5.0]),
])

all_students: list[Student] = [student1, student2]
```

b) (2 points) Implémentez la fonction suivante, qui doit, à partir de la liste de tous les élèves, retourner la moyenne des notes d'un élève (identifié par son ID) pour un cours donné (identifié par son nom). Votre code doit faire appel à la méthode `avg_grade()` de la classe `GradeRecord` de la partie a). Si l'élève ou le cours n'est pas trouvé, la fonction doit retourner `-1`.

```
def get_average_grade(all_students: list[Student], student_id: str, course_name: str) -> float:
```

c) (1 point) Sous la ligne suivante, écrivez du code qui, en parcourant la liste `all_students` et ses sous-structures, remplit le dictionnaire `indexed_avgs` de manière à ce qu'il relie chaque identifiant élève à un dictionnaire qui, lui, relie chaque nom de cours à la moyenne des notes de l'élève pour ce cours. Comme avant, faites appel à la méthode `avg_grade()`.
Après l'exécution de votre code, par exemple, `print(indexed_avgs["S001"]["Math"])` devra afficher `6.0`.

```
indexed_avgs: dict[str, dict[str, float]] =
```

Vous pouvez utiliser cette page au cas où vous n'auriez pas assez de place pour répondre à une question.
Pour faciliter la correction, **merci de bien indiquer** *à la question donnée qu'il faut venir lire la suite ici !*