Corrected

EPFL

Enseignant : Rafael Pires
CS-119(k) ICC Midterm -
13.04.2026 15h15
120 min.

0

Extra 1

SCIPER : **0** Salle : **CE 1 4** Signature :

Attendez le début de l'épreuve avant de tourner la page. Ce document est imprimé recto-verso, il contient 16 pages, les dernières pouvant être vides. Ne pas dégrafer.

- Posez votre carte d'étudiant sur la table.
- Une page A4 (recto-verso) de résumé personnel est autorisée.
- L'utilisation d'une **calculatrice** et de tout outil électronique est interdite pendant l'épreuve.
- Première partie **une seule bonne réponse possible (48 points)**:
  - On comptera :

    +3 points si la réponse est correcte,
    0 point si la réponse est incorrecte ou si il n'y a aucune ou plus d'une réponse inscrite,
- Deuxième partie **(10 points)** : Pour les questions **vrai/faux**, on comptera,:

  +1 points si la réponse est correcte,
  0 point si il n'y a aucune ou plus d'une réponse inscrite,
  −1 point si la réponse est incorrecte.
  ≥ 0 le total de la deuxième partie ne peut pas être négatif.
- Troisième partie **(12 points)** : pour les questions ouvertes, le nombre de point maximum est noté au-dessus de chaque question. Laissez les cases à cocher vides !
- Utilisez un **stylo** à encre **noire ou bleu foncé** et effacez proprement avec du **correcteur blanc** si nécessaire.
- Si une question est erronée, l'enseignant se réserve le droit de l'annuler.

Respectez les consignes suivantes | Observe this guidelines | Beachten Sie bitte die unten stehenden Richtlinien

choisir une réponse | select an answer
Antwort auswählen

ne PAS choisir une réponse | NOT select an answer
NICHT Antwort auswählen

Corriger une réponse | Correct an answer
Antwort korrigieren

ce qu'il ne faut **PAS** faire | what should **NOT** be done | was man **NICHT** tun sollte

Seules les réponses inscrites dans le cahier agrafé en français seront considérées ; les réponses sur le supplément de traduction anglaise, les brouillons ou feuilles volantes ne seront pas pris en compte.

Corrected

Première partie, questions à choix multiple

Pour chaque question marquer la/les case(s) correspondante(s) à la/aux réponse(s) correcte(s) sans faire de ratures.

Question 1

Que fait la fonction suivante ?

```python
def f(n:int):
    res = ''
    while n > 0:
        res = str(n % 2) + res
        n //= 2
    return res
```

- [x] Transforme l'entier n en binaire
- [ ] La boucle est infinie
- [ ] Calcule la somme des chiffres de n
- [ ] Calcule la somme des chiffres de l'écriture binaire de n

Explication

La boucle extrait le reste de la division par 2 (bit de poids faible) et le *préfixe* dans `res`, puis divise `n` par 2. C'est l'algorithme classique de conversion d'un entier en écriture binaire (lire les bits du LSB vers le MSB et les construire dans l'ordre inverse).

Question 2

Quel est le résultat affiché par le code Python suivant ?

```python
x = 7
print(f"{{x}} vaut {x}")
```

- [ ] `{x} vaut {x}`
- [x] `{x} vaut 7`
- [ ] `{{x}} vaut {{7}}`
- [ ] `x vaut 7`

Explication

Dans une f-string, `{{` et `}}` produisent des accolades littérales `{` et `}`, tandis que `{x}` est évalué et remplacé par la valeur de `x` (ici 7). Le résultat est donc `{x} vaut 7`.

Question 3

Quelle est la valeur affichée par le code Python suivant ?

```python
from dataclasses import dataclass

@dataclass
class Point:
    x: float
    y: float

    def distance_origine(self) -> float:
        return (self.x**2 + self.y**2) ** 0.5

p = Point(3.0, 4.0)
print(p.distance_origine())
```

☐ `7.0` ☐ `12.0` ☐ `25.0` ■ `5.0`

Explication

distance $= \sqrt{3{,}0^2 + 4{,}0^2} = \sqrt{9+16} = \sqrt{25} = 5{,}0$. C'est le triangle rectangle 3-4-5 de Pythagore.

Question 4

Laquelle de ces affirmations décrit le mieux un **algorithme glouton** ?

☐ Il mémoïse les résultats des sous-problèmes pour éviter les recalculs.

☐ Il divise le problème en deux sous-problèmes de taille égale, qu'il résout récursivement.

☐ Il explore toutes les solutions possibles et retourne la meilleure.

■ Il fait à chaque étape le choix localement optimal, sans jamais remettre en question les décisions prises.

Explication

Un algorithme **glouton** fait à chaque étape le choix localement optimal sans jamais revenir en arrière. Ce n'est pas un algorithme exhaustif (qui explore tout), ni de programmation dynamique (mémoïsation), ni diviser-pour-régner (division en deux moitiés).

Question 5

Quelle est la complexité temporelle (en notation Θ) de l'algorithme suivant, en fonction de n la longueur la liste L ?

```python
def f(L: list):
    S = 0
    k, m = 0, L[0]

    for i in range(1, len(L)):
        if L[i] > m:
            k, m = i, L[i]
    for j in range(k, len(L)):
        L[j] = m
```

■ $\Theta(n)$ ☐ $\Theta(n^2)$ ☐ $\Theta(\log(n))$ ☐ $\Theta(n \log n)$

Explication

La fonction contient deux boucles `for` **séquentielles** (non imbriquées), chacune de longueur au plus n. La première parcourt `range(1, n)` et la seconde `range(k, n)`. Le travail total est $\Theta(n) + \Theta(n) = \Theta(n)$.

Question 6

Quelle est la sortie du code Python suivant ?

```python
a = 17
b = 5
print(a // b, a % b)
```

- ☐ 3 3
- ☐ 3.4 2
- ☐ 2 3
- ■ 3 2

Explication

`17 // 5` est la division entière : $\lfloor 17/5 \rfloor = 3$. `17 % 5` est le reste : $17 - 5 \times 3 = 2$. Vérification : $5 \times 3 + 2 = 17$.

Question 7

Quelle est la bonne réponse ?

```python
def f(n: int) -> bool:
    s = str(n)
    rev = ""
    for c in s:
        rev = c + rev
    return s == rev
```

```python
def g(n: int) -> bool:
    s = str(n)

    if len(s) <= 1:
        return True
    if s[0] != s[-1]:
        return False

    return g(s[1:-1])
```

- ☐ Contrairement à f, la fonction g peut rester bloquée sans jamais finir.
- ■ Les deux fonctions retournent toujours le même résultat.
- ☐ Pour $n = 111111$, les deux fonctions retournent False.
- ☐ Pour $n = 123123$, f retourne False mais g retourne True.

Explication

`f` compare `s` à son inverse (construit itérativement) : elle teste si `n` est un palindrome. `g` fait de même récursivement en vérifiant que le premier et le dernier caractère sont égaux, puis en récursant sur la sous-chaîne intérieure. Les deux fonctions testent exactement la même propriété et retournent toujours le même résultat.

Question 8

Quelle est la sortie du code Python suivant ?

```python
a = [3, 2, 1, 2, 4]
b = set(a)
for j in range(0,len(a),2):
    b.add(j+2)
print(b)
```

- ☐ {1,2,2,3,4,4}
- ☐ {1,2,3,4,5}
- ☐ {3,4,5,6}
- ■ {1,2,3,4,6}

Explication

`b = set([3,2,1,2,4]) = {1,2,3,4}`. `range(0,5,2)` produit $j = 0, 2, 4$. Ajouts : $j = 0 \to$ ajoute 2 (déjà présent) ; $j = 2 \to$ ajoute 4 (déjà présent) ; $j = 4 \to$ ajoute 6 (nouveau). Résultat : $\{1, 2, 3, 4, 6\}$.

Question 9

Quelle est la sortie du code Python suivant ?

```python
def add_one(n: int) -> None:
    n = n + 1

def append_one(lst: list[int]) -> None:
    lst.append(1)

x = 10
my_list = [10]
add_one(x)
append_one(my_list)
print(x, my_list)
```

- ☐ `11 [10, 1]`
- ☐ `11 [10]`
- ■ `10 [10, 1]`
- ☐ `10 [10]`

Explication

Les entiers sont **immuables** en Python : `add_one` reçoit une copie de la valeur, la variable `x` reste `10`. Les listes sont **mutables** : `append_one` reçoit une référence vers le même objet en mémoire et le modifie en place, donc `my_list` devient `[10, 1]`.

Question 10

Quelle est la valeur décimale du nombre représenté en complément à deux sur 8 bits par `11101011` ?

- ☐ 20
- ☐ 235
- ■ -21
- ☐ -235

Explication

Le bit de signe vaut 1, donc le nombre est négatif. Méthode complément à deux : inverser tous les bits de `11101011` $\rightarrow$ `00010100`, puis ajouter 1 $\rightarrow$ `00010101` $= 16 + 4 + 1 = 21$. La valeur est donc -21.

Question 11

Quel est le contenu de `L` après l'exécution de ce code ?

```python
L = [10, 20, 30, 40, 50]
L[1:3] = [99]
```

- ☐ `[10, 20, 99, 40, 50]`
- ☐ `[10, 20, 99, 30, 40, 50]`
- ■ `[10, 99, 40, 50]`
- ☐ `[10, 99, 30, 40, 50]`

Explication

`L[1:3] = [99]` remplace les éléments aux indices 1 et 2 (valeurs 20 et 30) par le contenu de `[99]` (un seul élément). La liste passe de 5 à 4 éléments : `[10, 99, 40, 50]`.

Question 12

Qu'affiche le code suivant ?

```
d = {"a": 1, "b": 2}
for k in list(d.keys()):
    d[k+k] = d[k] * 2
print(len(d))
```

☐ Runtime error ☐ Boucle infinie ☐ 2 ■ 4

Explication

`list(d.keys())` crée un instantané `["a","b"]` avant la boucle, donc itérer ne cause pas d'erreur. Tour k=`"a"` : `d["aa"]=2` ; tour k=`"b"` : `d["bb"]=4`. Le dictionnaire final a 4 entrées : `"a"`, `"b"`, `"aa"`, `"bb"`.

Question 13

Qu'affiche le code suivant ?

```
def mystere(s):
    res = ""
    for i in range(len(s)):
        res = res + s[i] * (i + 1)
    return res
print(mystere("epfl"))
```

☐ eeppffffllll ☐ eppfll ☐ epfl ■ eppfffllll

Explication

Trace de `mystere("epfl")` : i=0 → `res="e"`, i=1 → `res="epp"`, i=2 → `res="eppfff"`, i=3 → `res="eppfffllll"`. Chaque caractère est répété $i+1$ fois.

Question 14

Quelle est la valeur affichée par le code Python suivant ?

```
def mystere(n: int) -> int:
    if n <= 1:
        return n
    return mystere(n - 1) + mystere(n - 2)

print(mystere(6))
```

☐ 5 ■ 8 ☐ 13 ☐ 6

Explication

`mystere` calcule la suite de Fibonacci (avec `mystere(0)=0`, `mystere(1)=1`). Les premières valeurs : 0, 1, 1, 2, 3, 5, **8**, 13, ... donc `mystere(6) = 8`.

Question 15

Quelle est la complexité temporelle (en notation Θ) de l'algorithme récursif **naïf** (sans mémoïsation) pour calculer le n-ieme terme de la suite de Fibonacci ?

☐ $\Theta(n)$ ■ $\Theta(2^n)$ ☐ $\Theta(n^2)$ ☐ $\Theta(n \log n)$

Explication

L'arbre des appels récursifs naïfs de Fibonacci a une profondeur n et chaque nœud engendre 2 appels fils. Le nombre de nœuds est de l'ordre de 2^n, d'où une complexité $\Theta(2^n)$.

Corrected

Question 16

Combien de comparaisons la recherche dichotomique effectue-t-elle **au maximum** pour chercher un élément dans une liste triée de 128 éléments ?

- ☐ 14
- ☐ 128
- ■ 7
- ☐ 64

Explication

$128 = 2^7$. La recherche dichotomique divise l'espace de recherche par 2 à chaque étape. Après k comparaisons, il reste $128/2^k$ éléments. On s'arrête quand $2^k \geq 128$, soit $k = \log_2(128) = 7$.

CORRECTED

Deuxième partie, questions du type Vrai ou Faux

Pour chaque question, marquer (sans faire de ratures) la case VRAI si l'affirmation est **toujours vraie** ou la case FAUX si elle **n'est pas toujours vraie** (c'est-à-dire si elle est parfois fausse).

Question 17 Un algorithme de complexité $\Theta(n^2)$ est **toujours** plus lent qu'un algorithme de complexité $\Theta(n \log n)$, quelle que soit la valeur de n.

☐ VRAI ■ FAUX

Explication

La notation Θ est **asymptotique** : elle décrit le comportement pour de grandes valeurs de n. Pour de petits n, n^2 peut être inférieur à $n \log n$ (par exemple $n = 1$: $1^2 = 1 < 0 = 1 \cdot \log 1$). Ce sont les constantes multiplicatives et la taille de n qui déterminent la vitesse réelle.

Question 18 Après l'exécution du code ci-dessous, la variable `s` contient la chaîne `"bon"`.

```
1 s = "bonjour"
2 s = s[:3]
```

■ VRAI ☐ FAUX

Explication

`"bonjour"[:3]` extrait les caractères aux indices 0, 1, 2 : `"b"`, `"o"`, `"n"`, soit la chaîne `"bon"`. Le slicing `[:3]` exclut l'indice 3.

Question 19 Après l'exécution du code suivant, la valeur 3 est affichée.

```
1 L1 = [1, 2, 3]
2 L2 = L1
3 L2[2] = 0
4 print(L1[2])
```

☐ VRAI ■ FAUX

Explication

`L2 = L1` ne copie pas la liste : les deux variables pointent vers le **même objet** en mémoire. Donc `L2[2] = 0` modifie aussi `L1[2]`, et `print(L1[2])` affiche `0`, pas `3`.

Question 20 La recherche dichotomique ne peut s'appliquer qu'à une liste préalablement triée.

■ VRAI ☐ FAUX

Explication

La recherche dichotomique exploite l'ordre de la liste : à chaque étape elle compare l'élément cherché à l'élément médian et élimine la moitié du tableau. Sans liste triée, on ne peut pas déterminer dans quelle moitié continuer.

CORRECTED

Question 21 La fonction `input()` en Python retourne toujours une valeur de type `str`, quelle que soit la valeur saisie par l'utilisateur.

■ VRAI ☐ FAUX

Explication

`input()` retourne **toujours** une chaîne de caractères (`str`), même si l'utilisateur tape `42` ou `3.14`. Pour obtenir un entier, il faut explicitement faire `int(input(...))`.

Question 22 En Python, un `set` peut contenir plusieurs fois la même valeur.

☐ VRAI ■ FAUX

Explication

Par définition, un `set` est un ensemble mathématique : il n'y a **pas de doublons**. `set([1,1,2,2])` donne `{1,2}`. Ajouter une valeur déjà présente ne change rien.

Question 23 Une clé de dictionnaire Python peut être de type `list`.

☐ VRAI ■ FAUX

Explication

Les clés d'un dictionnaire doivent être **hashables** (et donc immuables). Les `list` sont mutables et non-hashables : tenter `d[[1,2]] = "x"` lève une `TypeError`. Il faut utiliser un `tuple` à la place.

Question 24 La mémoïsation de l'algorithme récursif naïf de Fibonacci réduit sa complexité temporelle à $\Theta(n)$.

■ VRAI ☐ FAUX

Explication

Sans mémoïsation, Fibonacci récursif recalcule les mêmes sous-problèmes exponentiellement ($\Theta(2^n)$). Avec mémoïsation, chaque valeur $F(0), F(1), \ldots, F(n)$ est calculée **une seule fois**, donc n calculs au total : $\Theta(n)$.

Question 25 Le problème de l'arrêt est indécidable : il n'existe aucun algorithme capable de déterminer, pour tout programme P et toute entrée X, si $P(X)$ s'arrête.

■ VRAI ☐ FAUX

Explication

Résultat fondamental de l'informatique théorique démontré par Alan Turing en 1936 par diagonalisation. Si un tel algorithme existait, on pourrait construire un programme P^* dont le comportement serait contradictoire, d'où l'impossibilité.

Question 26 Une fonction Python qui ne contient aucune instruction `return` retourne implicitement le type `None`.

■ VRAI ☐ FAUX

Explication

En Python, toute fonction qui atteint la fin de son corps sans rencontrer `return` (ou qui exécute `return` sans valeur) retourne implicitement `None`.

Troisième 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 27: *Cette question est notée sur 6 points.*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ■6

On construit un petit système de gestion d'un catalogue de films. Considérez le code suivant :

```python
from dataclasses import dataclass

@dataclass
class Film:
    titre: str
    realisateur: str
    annee: int
    notes: list[float]  # notes attribuees par les critiques (entre 0 et 10)
```

a) (2 point) Ajoutez à la classe `Film` une méthode `note_moyenne()` qui retourne la moyenne des notes. Si la liste de notes est vide, la méthode doit retourner `-1.0`.

```python
@dataclass
class Film:
    titre: str
    realisateur: str
    annee: int
    notes: list[float]
```

Solution

```python
    def note_moyenne(self) -> float:
        if len(self.notes) == 0:
            return -1.0
        return sum(self.notes) / len(self.notes)
```

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

```python
catalogue: list[Film] = [
    Film("Metropolis",  "Fritz Lang",     1927, [8.3, 9.1, 7.8]),
    Film("Vertigo",     "Alfred Hitchcock", 1958, [9.0, 8.5]),
    Film("Psycho",      "Alfred Hitchcock", 1960, [8.8, 9.2, 8.6]),
    Film("Alien",       "Ridley Scott",   1979, [8.4, 7.9, 8.7]),
    Film("Blade Runner","Ridley Scott",   1982, []),
    Film("Tenet",       "Christopher Nolan", 2020, [6.5, 7.0]),
]
```

b) (2 points) Implémentez la fonction suivante, qui retourne la liste des films réalisés par `realisateur` (identifié par son nom), triés par ordre croissant d'année. Si aucun film n'est trouvé, la fonction doit retourner une liste vide.

Indice : la fonction `sorted(liste, key=...)` peut trier une liste selon un critère de votre choix.

```python
def films_par_realisateur(catalogue: list[Film],
                          realisateur: str) -> list[Film]:
```

Solution

```python
def films_par_realisateur(catalogue: list[Film],
                          realisateur: str) -> list[Film]:
    films = []
    for film in catalogue:
        if film.realisateur == realisateur:
            films.append(film)
    films.sort(key=lambda film: film.annee)
    return films
```

c) (2 points) Sous la ligne suivante, écrivez du code qui, en parcourant `catalogue` et en faisant appel à la méthode `note_moyenne()` de la partie a), construit le dictionnaire `index_notes`. Ce dictionnaire doit associer à chaque `titre` de film sa note moyenne. Les films dont la liste de notes est vide ne doivent **pas** apparaître dans le dictionnaire.

Après l'exécution de votre code, par exemple, `print(index_notes["Vertigo"])` devra afficher `8.75`.

```python
index_notes: dict[str, float] = {}
```

Solution

```python
for film in catalogue:
    moyenne = film.note_moyenne()
    if moyenne != -1.0:
        index_notes[film.titre] = moyenne
```

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

CORRECTED

Question 28: *Cette question est notée sur 6 points.*

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ■6

Soit x un entier non nul et soit p un entier positif. On cherche à calculer x^p et on considère que la multiplication de deux entiers se fait en $\Theta(1)$

a) (1 point) Quelle est la compléxité du calcul naïf de x^p?

Solution

La complexité est $\Theta(p)$.

b) (2 points) En remarquant que

$$\begin{aligned}
&\text{Si } p = 0, \text{alors } x^p = 1\\
&\text{Si } p = 2k \text{ est pair, alors } x^p = (x^k)^2\\
&\text{Si } p = 2k+1 \text{ est impair, alors } x^p = x \times (x^k)^2
\end{aligned}$$

Ecrivez une fonction **récursive** qui effectue le calcul de x^p en $\Theta(\log_2(p))$ opérations.

```python
def recursive_puissance(x:int, p:int):
```

Solution

```python
def puissance(x: float, p: int) -> float:
    if p == 0:
        return 1
    elif p % 2 == 0:
        y = puissance(x, p // 2)
        return y * y
    else:
        return x * puissance(x, p - 1)
```

c) (1 point)

Etant données 2 matrices $A = \begin{pmatrix} a1 & b1 \\ c1 & d1 \end{pmatrix}$ et $B = \begin{pmatrix} a2 & b2 \\ c2 & d2 \end{pmatrix}$, représentées sous forme de liste de listes en Python, comme ci-dessous

```
A = [[a1, b1], [c1, d1]]
B = [[a2, b2], [c2, d2]]
```

ecrivez une fonction Python qui retourne le produit matriciel des deux matrices et calculer sa complexité exacte en nombre de multiplications et d'additions.

```
def multiply(A : list[list], B : list[list]) -> list[list]:
```

Solution

```
def multiply(A, B):
    return [
        [
            A[0][0] * B[0][0] + A[0][1] * B[1][0],
            A[0][0] * B[0][1] + A[0][1] * B[1][1]
        ],
        [
            A[1][0] * B[0][0] + A[1][1] * B[1][0],
            A[1][0] * B[0][1] + A[1][1] * B[1][1]
        ]
    ]
```

Complexité : 8 multiplications et 4 additions, soit $\Theta(1)$.

d) (2 points)

On considère maintenant la matrice $M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$, et on note M^n le produit de M par elle-même n fois. Il est possible de montrer par récurrence que pour tout $n \geq 1$:

$$M^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

En admettant ce résultat **(sans le démontrer)**, et en utilisant la fonction **multiply** de la question précédente, proposer un algorithm qui calcul le n-ième terme de la suite de Fibonacci en $\Theta(\log_2(n))$ opérations.

Solution

```python
def puissance_matrice(M, n):
    if n == 1:
        return M
    if n % 2 == 0:
        half = puissance_matrice(M, n // 2)
        return multiply(half, half)
    else:
        return multiply(M, puissance_matrice(M, n - 1))

def fibonacci(n):
    if n == 0:
        return 0
    M = [[1, 1], [1, 0]]
    Mn = puissance_matrice(M, n)
    return Mn[0][1]
```

Place supplémentaire pour répondre à n'importe quelle question si nécessaire. Mais VEUILLEZ INDIQUER LE NUMÉRO DE LA QUESTION TRAITÉE.