NSI Terminale — Architectures matérielles, systèmes d'exploitation et réseaux

Sujet d'entraînement

Durée : 1 h 50 · Calculatrice interdite · Le sujet comporte quatre exercices indépendants. Total sur 20 points.

Nom : ..... Prénom : ..... Classe : .....

Les réponses sont à rédiger sur la copie. Les programmes Python demandés s'écrivent à la main ; une petite erreur de syntaxe n'est pas pénalisée si l'idée est juste.

Exercice 1 — Les systèmes sur puce (2,5 points)

1. Un smartphone récent est construit autour d'un système sur puce (SoC). (1,5 point)

a) Expliquer ce qu'est un système sur puce.

b) Recopier et compléter le tableau en indiquant le rôle de chaque bloc.

Bloc	Rôle
CPU	
GPU	
Gestion d'énergie	
Réseau sur puce (bus)	

c) Citer deux avantages et un inconvénient de l'intégration sur une seule puce.

d) Expliquer pourquoi un smartphone n'a pas besoin de ventilateur, et pourquoi on ne peut pas lui ajouter de mémoire vive.

2. En 1989, le processeur Intel 486 comptait environ 1,2 million de transistors. On suppose que ce nombre double exactement tous les deux ans. (1 point)

a) Combien de doublements y a-t-il entre 1989 et 2009 ?

b) En utilisant $2^{10} \approx 10^3$, donner l'ordre de grandeur du nombre de transistors prévu pour 2009.

c) La loi de Moore est-elle une loi physique ? Pourquoi ne peut-elle pas se prolonger indéfiniment ?

Exercice 2 — Processus, ordonnancement et interblocage (7 points)

1. Tom obtient l'affichage suivant avec la commande `ps -ef` (seules quatre colonnes sont conservées). (1,5 point)

UID	PID	PPID	CMD
root	1	0	/sbin/init
root	612	1	/usr/sbin/cupsd
tom	1980	1	gnome-terminal
tom	2150	1980	bash
tom	2151	1980	bash
tom	2203	2150	python3 serveur.py
tom	2204	2151	python3 client.py
tom	2290	2151	nano notes.txt

a) Quel est le PPID du processus `python3 client.py` ? Quel programme exécute son parent ?

b) Dessiner l'arbre des processus.

c) Le programme `client.py` ne répond plus. Écrire la commande qui permet de le terminer.

d) Expliquer, à l'aide de cet affichage, la différence entre un programme et un processus.

2. Les états d'un processus. (1,5 point)

a) Le processus `serveur.py` est créé, attend son tour, s'exécute, attend une requête du réseau, la reçoit, attend son tour, s'exécute à nouveau, puis se termine. Donner la suite des états par lesquels il passe.

b) Pourquoi un processus bloqué ne passe-t-il jamais directement à l'état élu ?

On représente les transitions autorisées par un dictionnaire : à chaque état, on associe la liste des états dans lesquels le processus peut passer ensuite.

```
TRANSITIONS = {
    "Nouveau": ["Prêt"],
    "Prêt": ["Élu"],
    "Élu": ......................,
    "Bloqué": ......................,
    "Terminé": []
}

def suite_valide(etats: list) -> bool:
    """Renvoie True si chaque passage d'un état au
    suivant est autorisé."""
    for i in range(len(etats) - 1):
        if ..........................................:
            return False
    return True
```

c) Recopier et compléter les entrées `"Élu"` et `"Bloqué"`.

d) Recopier et compléter la condition de la fonction `suite_valide`.

e) Que renvoie la fonction pour la suite d'états obtenue à la question a) ? Pour la suite `["Nouveau", "Élu", "Terminé"]` ? Justifier.

3. Cinq processus arrivent dans le système (dates et durées en unités de temps). (2 points)

Processus	P1	P2	P3	P4	P5
Date d'arrivée	0	1	2	3	6
Durée	4	6	2	1	3

Le **temps de séjour** d'un processus vaut : date de fin − date d'arrivée. Pour le tourniquet, un processus qui arrive à l'instant où un quantum se termine est placé dans la file **avant** le processus interrompu. Pour « plus court d'abord », chaque fois que le processeur se libère, on élit, parmi les processus arrivés, celui de plus courte durée, qui s'exécute jusqu'au bout. Chaque frise est à recopier sur la copie ; la case k va de l'instant k à l'instant k + 1.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Processus élu																

a) Compléter une frise pour la politique « premier arrivé, premier servi » (FIFO).

b) Compléter une frise pour la politique « plus court d'abord ».

c) Compléter une frise pour le tourniquet avec un quantum de 2 unités de temps. Préciser l'état de la file à l'instant 4.

d) Pour chaque politique, donner les dates de fin et calculer le temps de séjour moyen.

e) Quelle politique minimise le temps de séjour moyen ? Quel est son défaut ? Laquelle des trois est préemptive, et que signifie ce mot ? Pourquoi est-ce elle qu'on utilise sur un ordinateur interactif ?

4. La fonction suivante calcule les dates de fin pour la politique FIFO. (0,5 point)

```
def dates_fin(processus: list) -> dict:
    """processus : liste de tuples (nom, arrivee, duree),
    triee par date d'arrivee."""
    t = 0
    fins = {}
    for nom, arrivee, duree in processus:
        if t < arrivee:
            t = .........        # ligne A
        t = .........            # ligne B
        fins[nom] = t
    return fins
```

Recopier et compléter les lignes A et B. Que renvoie la fonction pour les cinq processus de la question 3 ? À quoi correspond le cas `t < arrivee` ?

5. Cinq processus utilisent quatre ressources R1 à R4, chacune disponible en un seul exemplaire. (1,5 point)

Processus	Détient	Attend
P1	R1	R2
P2	R2	R3
P3	R3	rien
P4	R4	R1
P5	rien	R2

a) Dessiner le graphe d'attente. Y a-t-il interblocage ? Justifier.

b) Si rien ne change, dans quel ordre les processus peuvent-ils terminer ?

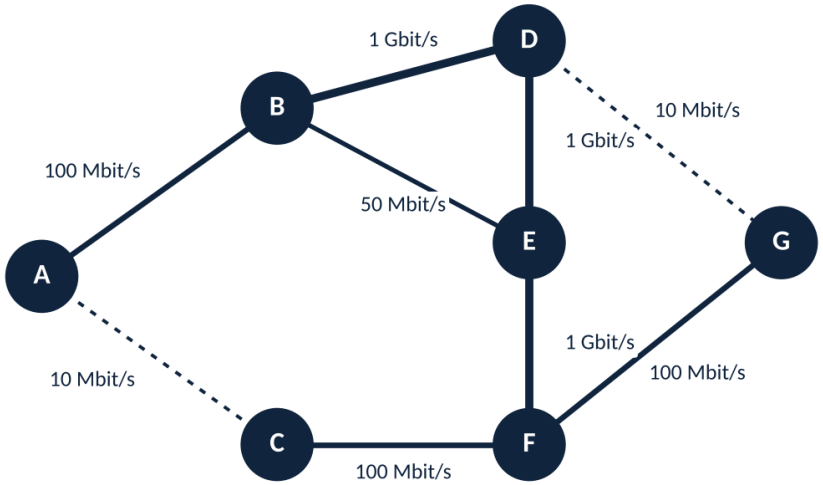
c) En réalité, avant de terminer, P3 demande la ressource R4. Quels processus sont alors en interblocage ? Quel processus est bloqué sans faire partie de l'interblocage ?

d) Le système décide d'arrêter un processus. Arrêter P5 suffit-il ? Lesquels peut-il arrêter ?

e) On impose l'ordre R1, R2, R3, R4 : un processus ne peut demander qu'une ressource placée après toutes celles qu'il détient. Quel processus ne respecte pas cette règle ? Expliquer pourquoi, si tous la respectent, aucun cycle ne peut se former.

Exercice 3 — Routage (6,5 points)

Une entreprise relie sept routeurs A à G. Le schéma indique le débit de chaque liaison.



1. Le protocole RIP. Les échanges sont simultanés : à chaque échange, chaque routeur envoie à ses voisins la table qu'il avait à la fin de l'échange précédent. Un routeur lit les tables reçues dans l'ordre alphabétique des voisins, et ne remplace une route que par une route **strictement** plus courte. (2,5 points)

- **a)** Que minimise le protocole RIP ?
- **b)** Écrire la table de A au démarrage (destination, passerelle, distance).
- **c)** Écrire la table de A après le premier échange.
- **d)** Écrire la table de A après le deuxième échange. Expliquer le choix de la passerelle vers G.
- **e)** Combien d'échanges faut-il pour que la table de A ne change plus ? À partir de quel échange A le constate-t-il ?
- **f)** Sachant que B transmet à D les paquets destinés à G, donner le chemin complet suivi de A vers G avec RIP.

2. Le protocole OSPF. Le coût d'une liaison de débit d, en bit/s, vaut $\frac{10^8}{d}$. **On conserve les décimales.** (2,5 points)

- **a)** Calculer le coût d'une liaison à 1 Gbit/s, 100 Mbit/s, 50 Mbit/s et 10 Mbit/s.
- **b)** Calculer le coût des chemins A → B → D → G, A → C → F → G, A → B → E → F → G et A → B → D → E → F → G. Quel chemin OSPF choisit-il ? Combien de sauts compte-t-il ?
- **c)** La liaison E–F tombe en panne. Quel est le nouveau chemin OSPF de A vers G, et son coût ?
- **d)** Expliquer pourquoi le chemin choisi par RIP à la question 1 f) est un mauvais choix.

3. Programmation. (1,5 point)

```
tables = {"A": {"G": "B"}, "B": {"G": "D"}, "D": {"G": "G"}}

couts = {("A", "B"): 1, ("A", "C"): 10, ("B", "D"): 0.1,
         ("B", "E"): 2, ("C", "F"): 1, ("D", "E"): 0.1,
         ("D", "G"): 10, ("E", "F"): 0.1, ("F", "G"): 1}

def route(tables: dict, depart: str, arrivee: str) -> list:
    chemin = [depart]
    routeur = depart
    while routeur != arrivee:
        routeur = ........................
        chemin.append(.........)
    return chemin

def cout(x: str, y: str) -> float:
    if (x, y) in couts:
        return couts[(x, y)]
    return ...................

def cout_chemin(chemin: list) -> float:
    total = 0
    for i in range(...................):
        total = total + cout(........., .........)
    return total
```

- **a)** Recopier et compléter la fonction `route`. Que renvoie `route(tables, "A", "G")` ?
- **b)** Recopier et compléter les fonctions `cout` et `cout_chemin`. Pourquoi `cout` envisage-t-elle les deux ordres ?
- **c)** Que renvoie `cout_chemin(["G", "D", "B", "A"])` ?
- **d)** Un élève exécute `cout_chemin(["A", "B", "D", "E", "F", "G"])` et obtient `2.3000000000000003`. Expliquer ce résultat.

Exercice 4 — Sécurisation des communications (4 points)

1. Chiffrement symétrique par XOR. (0,75 point)

- **a)** Chiffrer l'octet `01010011` avec la clé `11001010` par un OU exclusif bit à bit.
- **b)** Déchiffrer le résultat. Expliquer pourquoi refaire un XOR avec la même clé redonne le message.

2. Attaque par force brute. Un attaquant teste un million (10^6) de clés par seconde. (1 point)

a) Combien existe-t-il de clés de 8 bits ? Pourquoi une telle clé n'offre-t-elle aucune protection ?

b) On admet que $2^{32} \approx 4{,}3 \times 10^9$. Combien de temps faut-il, au plus, pour essayer toutes les clés de 32 bits ? Donner le résultat en heures et minutes.

c) Que devient ce temps si on ajoute un seul bit à la clé ? Pourquoi les clés de 128 bits sont-elles sûres ?

3. Chiffrement asymétrique. (1 point)

a) Inès veut envoyer un message secret à Yanis. Quelle clé utilise-t-elle pour chiffrer, et quelle clé Yanis utilise-t-il pour déchiffrer ? Ève connaît la clé publique de Yanis : peut-elle lire le message ?

b) Comparer les chiffrements symétrique et asymétrique (clés, vitesse, point faible). En déduire pourquoi HTTPS combine les deux.

4. Le protocole HTTPS. (1,25 point)

a) Remettre ces étapes dans l'ordre :

(1) Le serveur déchiffre la clé de session avec sa clé privée.

(2) Le navigateur contrôle la signature du certificat.

(3) Le navigateur et le serveur échangent des données chiffrées en symétrique.

(4) Le serveur transmet son certificat, qui contient sa clé publique.

(5) Le navigateur demande une connexion sécurisée.

(6) Le navigateur crée une clé de session et l'envoie chiffrée avec la clé publique du serveur.

b) Un attaquant placé entre le navigateur et le serveur remplace la clé publique du serveur par la sienne. Que pourrait-il faire ensuite ? Quelle étape l'en empêche ?

Sujet d'entraînement

Corrigé détaillé et barème — document réservé au professeur

Question	Points	Attendus et répartition
Ex. 1 – Q1	1,5	a) 0,25. b) 0,5 : 0,125 par bloc. c) 0,5 : deux avantages + un inconvénient. d) 0,25.
Ex. 1 – Q2	1	a) 0,25 : 10. b) 0,5 : environ 1 milliard. c) 0,25 : observation empirique + limite.
Ex. 2 – Q1	1,5	a) 0,25. b) 0,75. c) 0,25. d) 0,25 : deux processus bash pour un même programme.
Ex. 2 – Q2	1,5	a) 0,5. b) 0,25. c) 0,25. d) 0,25. e) 0,25 : True et False justifiés.
Ex. 2 – Q3	2	a) 0,25. b) 0,5. c) 0,5 (dont état de la file). d) 0,5. e) 0,25.
Ex. 2 – Q4	0,5	Lignes A et B (0,25) ; résultat et processeur inoccupé (0,25).
Ex. 2 – Q5	1,5	a) 0,25. b) 0,25. c) 0,5. d) 0,25. e) 0,25.
Ex. 3 – Q1	2,5	a) 0,25. b) 0,25. c) 0,5. d) 0,5 (dont règle d'égalité). e) 0,5. f) 0,5.
Ex. 3 – Q2	2,5	a) 0,5. b) 1 : quatre coûts + choix + sauts. c) 0,5. d) 0,5.
Ex. 3 – Q3	1,5	a) 0,5. b) 0,5. c) 0,25. d) 0,25.
Ex. 4 – Q1	0,75	a) 0,25. b) 0,5 : résultat + justification.
Ex. 4 – Q2	1	a) 0,25. b) 0,5. c) 0,25.
Ex. 4 – Q3	1	a) 0,5. b) 0,5.
Ex. 4 – Q4	1,25	a) 0,75 : ordre exact (0,25 si une seule inversion). b) 0,5.
Total	**20**	

Exercice 1 (2,5 points)

1. Systèmes sur puce (1,5 point)

a) Un système sur puce est **un circuit intégré unique qui regroupe les composants d'un ordinateur**, autrefois répartis sur plusieurs circuits assemblés sur une carte.

b) Rôle des blocs :

Bloc	Rôle
CPU	Exécute les instructions des programmes (plusieurs cœurs).
GPU	Calcule l'affichage et traite les images.
Gestion d'énergie	Régule l'alimentation de chaque bloc et éteint ceux qui ne servent pas.
Réseau sur puce (bus)	Relie les blocs entre eux, à des débits adaptés à chacun.

c) **Avantages** (deux parmi) : vitesse, consommation réduite, faible encombrement, coût réduit en grande série.

Inconvénient : difficile à réparer, impossible à faire évoluer.

d) Les composants, très proches, échangent leurs signaux sur de courtes distances : **peu d'énergie perdue en chaleur**, et la gestion d'énergie éteint les blocs inutiles. La mémoire est **intégrée à la puce** ou soudée contre elle : on ne peut pas en ajouter.

2. Loi de Moore (1 point)

a) $\frac{2009-1989}{2}=\frac{20}{2}=10$

10 doublements.

b) $1{,}2\times10^{6}\times2^{10}\approx1{,}2\times10^{6}\times10^{3}$

$1{,}2\times10^{6}\times10^{3}=1{,}2\times10^{9}$

Environ 1 milliard de transistors.

c) Non : c'est **une observation empirique**, pas une loi physique.

Elle ne peut pas durer indéfiniment : les transistors approchent la taille de quelques atomes, et la chaleur produite devient difficile à évacuer.

Exercice 2 (7 points)

1. La commande ps (1,5 point)

a) Le PPID de `python3 client.py` est **2151** : son parent exécute **bash**.

b) Arbre des processus :

```
1 /sbin/init
├── 612 /usr/sbin/cupsd
└── 1980 gnome-terminal
    ├── 2150 bash
    │   └── 2203 python3 serveur.py
    └── 2151 bash
        ├── 2204 python3 client.py
        └── 2290 nano notes.txt
```

c) **kill 2204**

d) Un **programme** est un fichier ; un **processus** est un programme en cours d'exécution. Ici, le même programme `bash` a donné **deux processus distincts**, 2150 et 2151, chacun avec son PID, sa mémoire et ses enfants.

2. États et transitions (1,5 point)

a) **Nouveau → Prêt → Élu → Bloqué → Prêt → Élu → Terminé.**

b) Quand l'événement attendu arrive, le processeur est peut-être occupé par un autre processus : le processus redevient **prêt**, et c'est l'**ordonnanceur** qui décide quand il sera élu.

c) Entrées complétées :

```
    "Élu": ["Prêt", "Bloqué", "Terminé"],
    "Bloqué": ["Prêt"],
```

d) Condition complétée :

```
        if etats[i + 1] not in TRANSITIONS[etats[i]]:
```

e) Pour la suite de la question a), la fonction renvoie **True** : chaque passage est autorisé.

Pour `["Nouveau", "Élu", "Terminé"]`, elle renvoie **False** : `TRANSITIONS["Nouveau"]` vaut `["Prêt"]`, et « Élu » n'y figure pas.

3. Ordonnancement (2 points)

a) **FIFO** :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FIFO	P1	P1	P1	P1	P2	P2	P2	P2	P2	P2	P3	P3	P4	P5	P5	P5

b) **Plus court d'abord** :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Plus court	P1	P1	P1	P1	P4	P3	P3	P5	P5	P5	P2	P2	P2	P2	P2	P2

À l'instant 4, P2 (6), P3 (2) et P4 (1) attendent : on élit P4, puis P3 à l'instant 5. À l'instant 7, P2 (6) et P5 (3) attendent : on élit P5, puis P2.

c) **Tourniquet, quantum 2** :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Tourniquet	P1	P1	P2	P2	P3	P3	P1	P1	P4	P2	P2	P5	P5	P2	P2	P5

À l'instant 2, P1 est interrompu ; P2 et P3 sont déjà dans la file, P1 se place derrière eux.

À l'instant 4, P2 est interrompu et P4 vient d'arriver (en 3) : P4 entre dans la file avant P2. **La file est P3, P1, P4, P2.**

Suite de l'exécution : P3 (4 à 6), P1 (6 à 8), P4 (8 à 9), P2 (9 à 11), P5 (11 à 13), P2 (13 à 15), P5 (15 à 16).

d) Dates de fin et temps de séjour :

	P1	P2	P3	P4	P5
Fin FIFO	4	10	12	13	16
Séjour FIFO	4	9	10	10	10
Fin plus court	4	16	7	5	10
Séjour plus court	4	15	5	2	4
Fin tourniquet	8	15	6	9	16
Séjour tourniquet	8	14	4	6	10

Moyenne FIFO :

$$\frac{4+9+10+10+10}{5}=\frac{43}{5}=8{,}6$$

Moyenne « plus court d'abord » :

$$\frac{4+15+5+2+4}{5}=\frac{30}{5}=6$$

Moyenne tourniquet :

$$\frac{8+14+4+6+10}{5}=\frac{42}{5}=8{,}4$$

FIFO : 8,6 ; plus court d'abord : 6 ; tourniquet : 8,4.

e) **« Plus court d'abord » minimise le temps de séjour moyen**, mais un processus long (ici P2) peut attendre indéfiniment si des processus courts arrivent sans cesse : c'est la **famine**.

Le **tourniquet est préemptif** : l'ordonnanceur peut retirer le processeur à un processus avant qu'il ait terminé (à la fin de son quantum).

Sur un ordinateur interactif, chaque application reçoit régulièrement le processeur : aucune ne reste figée derrière un long calcul, ce qui donne l'impression que tout s'exécute en même temps. Avec le tourniquet, P3 et P4 terminent ici bien plus tôt qu'en FIFO.

4. Fonction dates_fin (0,5 point)

```
        if t < arrivee:
            t = arrivee          # ligne A
        t = t + duree            # ligne B
```

La fonction renvoie **{"P1": 4, "P2": 10, "P3": 12, "P4": 13, "P5": 16}**, les dates de fin FIFO de la question 3.

Le cas `t < arrivee` correspond à un **processeur inoccupé** : aucun processus n'est encore arrivé. Il ne se produit pas avec ces données.

5. Interblocage (1,5 point)

a) Graphe : R1 → P1 → R2 → P2 → R3 → P3 ; R4 → P4 → R1 ; P5 → R2 (processus → ressource : « attend » ; ressource → processus : « est détenue par »).

P3 n'attend rien : **pas de cycle, donc pas d'interblocage**.

b) **P3** termine et libère R3 ; **P2** obtient R3, termine et libère R2 et R3 ; **P1** obtient R2, termine et libère R1 et R2 ; **P4** obtient R1 et **P5** obtient R2 : ils peuvent terminer à leur tour.

P5 peut aussi obtenir R2 avant P1 ; P1 l'obtiendra ensuite. Toute réponse cohérente est acceptée.

c) On ajoute P3 → R4. Le cycle **P1 → R2 → P2 → R3 → P3 → R4 → P4 → R1 → P1** apparaît : **P1, P2, P3 et P4 sont en interblocage.**

P5 est bloqué (il attend R2, détenue par P2) **sans faire partie du cycle**.

d) **Arrêter P5 ne suffit pas** : il ne détient aucune ressource. Il faut arrêter **l'un des processus du cycle : P1, P2, P3 ou P4**.

e) **P4** détient R4 et demande R1, placée avant : il ne respecte pas la règle. (P3, qui détient R3 et demande R4, la respecte.)

Si tous respectent la règle, un processus n'attend jamais qu'une ressource de rang plus élevé que toutes celles qu'il détient. En suivant un cycle, les

rangs des ressources augmenteraient strictement à chaque étape, et l'on reviendrait pourtant à la ressource de départ : c'est impossible. **Aucun cycle ne peut se former.**

Exercice 3 (6,5 points)

1. RIP (2,5 points)

a) RIP minimise **le nombre de sauts** (routeurs traversés).

b) Au démarrage :

Destination	Passerelle	Distance
B	B	1
C	C	1

c) Premier échange : B annonce D et E à 1 saut, C annonce F à 1 saut.

Destination	Passerelle	Distance
B	B	1
C	C	1
D	B	2
E	B	2
F	C	2

d) Pendant le premier échange, B a appris G à 2 sauts (par D), et C a appris G à 2 sauts (par F). Au deuxième échange, A lit d'abord la table de B : il ajoute G à 3 sauts par B. Puis il lit celle de C, qui propose aussi G à 3 sauts : **ce n'est pas strictement plus court, A garde la passerelle B.**

Destination	Passerelle	Distance
B	B	1
C	C	1
D	B	2
E	B	2
F	C	2
G	B	3

e) La table de A change aux échanges 1 et 2 seulement. **Il faut deux échanges ; A le constate au troisième**, qui n'apporte plus rien.

f) A envoie à B, B envoie à D, et D est relié à G : **A → B → D → G** (3 sauts).

2. OSPF (2,5 points)

a) Coûts :

1 Gbit/s : $\frac{10^8}{10^9} = 0{,}1$

100 Mbit/s : $\frac{10^8}{10^8} = 1$

50 Mbit/s : $\frac{10^8}{5 \times 10^7} = 2$

10 Mbit/s : $\frac{10^8}{10^7} = 10$

b) Coût des chemins :

A → B → D → G : $1+0{,}1+10 = 11{,}1$

A → C → F → G : $10+1+1 = 12$

$A \to B \to E \to F \to G : 1+2+0{,}1+1=4{,}1$

$A \to B \to D \to E \to F \to G : 1+0{,}1+0{,}1+0{,}1+1=2{,}3$

OSPF choisit A → B → D → E → F → G, de coût 2,3, soit 5 sauts.

c) Sans E–F, il reste notamment A → B → D → G (11,1), A → C → F → G (12) et A → B → E → D → G :

$1+2+0{,}1+10=13{,}1$

Nouveau chemin : A → B → D → G, de coût 11,1.

d) Le chemin RIP A → B → D → G emprunte la liaison D–G à 10 Mbit/s, la plus lente : son coût vaut 11,1, presque cinq fois plus que le chemin OSPF (2,3). RIP **ne tient pas compte du débit**.

3. Programmation (1,5 point)

a) Fonction route :

```
        routeur = tables[routeur][arrivee]
        chemin.append(routeur)
```

`route(tables, "A", "G")` renvoie **["A", "B", "D", "G"]**.

b) Fonctions cout et cout_chemin :

```
    return couts[(y, x)]

    for i in range(len(chemin) - 1):
        total = total + cout(chemin[i], chemin[i + 1])
```

Chaque liaison n'est mémorisée qu'une fois, mais elle s'emprunte dans les deux sens : ("D", "B") n'est pas une clé, il faut chercher ("B", "D").

c) $10+0{,}1+1=11{,}1$

La fonction renvoie 11.1.

d) Le nombre 0,1 **n'a pas d'écriture binaire finie** : l'ordinateur en stocke une valeur approchée. En additionnant plusieurs de ces approximations, l'erreur devient visible : on obtient 2.3000000000000003 au lieu de 2,3.

Conséquence pratique : on ne compare jamais deux flottants avec ==.

Exercice 4 (4 points)

1. XOR (0,75 point)

a) Chiffrement :

	Octet
Message	01010011
Clé	11001010
Chiffré	**10011001**

b) Déchiffrement :

	Octet
Chiffré	10011001
Clé	11001010
Message	**01010011**

Pour chaque bit, $b \oplus k \oplus k = b \oplus 0 = b$: un bit combiné deux fois avec le même bit de clé redevient lui-même.

2. Force brute (1 point)

a) $2^8 = 256$

256 clés seulement : un programme les essaie toutes en une fraction de seconde.

b) $\dfrac{4{,}3 \times 10^9}{10^6} = 4\,300$

$4\,300 = 3\,600 + 700$, et $700 \approx 11{,}7 \times 60$.

Environ 4 300 s, soit 1 h 12 min.

c) Un bit de plus **double** le nombre de clés, donc le temps : environ 2 h 24 min.

Avec 128 bits, il y a 2^{128} clés : même avec des milliards de machines, il faudrait bien plus que l'âge de l'Univers.

3. Chiffrement asymétrique (1 point)

a) Inès chiffre avec **la clé publique de Yanis** ; Yanis déchiffre avec **sa clé privée**. Ève ne peut pas lire le message : la clé publique permet seulement de chiffrer.

b) Comparaison :

Critère	Symétrique	Asymétrique
Clés	Une clé secrète partagée	Une paire par personne
Vitesse	Rapide	Beaucoup plus lent
Point faible	Échanger la clé en secret	Trop lent pour de gros volumes

HTTPS utilise l'asymétrique pour **échanger une clé de session** sans rendez-vous préalable, puis le **symétrique, rapide**, pour toute la communication.

4. HTTPS (1,25 point)

a) **Ordre : 5 → 4 → 2 → 6 → 1 → 3.**

b) Le navigateur chiffrerait la clé de session avec la clé de l'attaquant : celui-ci la **déchiffrerait avec sa clé privée** et pourrait lire, voire modifier, toute la communication.

C'est l'**étape 2** qui l'en empêche : le certificat est **signé par une autorité de certification**, et le navigateur vérifie cette signature. L'attaquant ne peut pas fournir de certificat valide associant le nom du site à sa propre clé.