Ce document a été mis en ligne par l'organisme FormaV®.

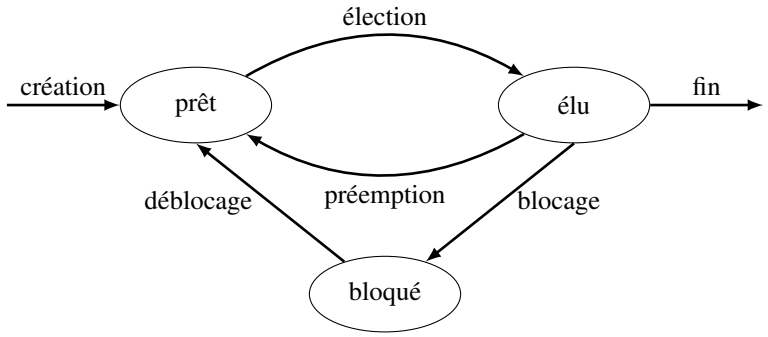
Toute reproduction, représentation ou diffusion, même partielle, sans autorisation préalable, est strictement interdite.

Pour consulter tous nos sujets d'annales et corrigés du Bac Général Maths + NSI, veuillez visiter :
bac-maths-nsi.fr/annales-sujets-et-corriges

Amérique du nord - mai 2026 - sujet 2 (corrigé)

Exercice 1 (Architecture matérielle, OS et structures de données linéaires)
Partie A

1. On a le schéma suivant :

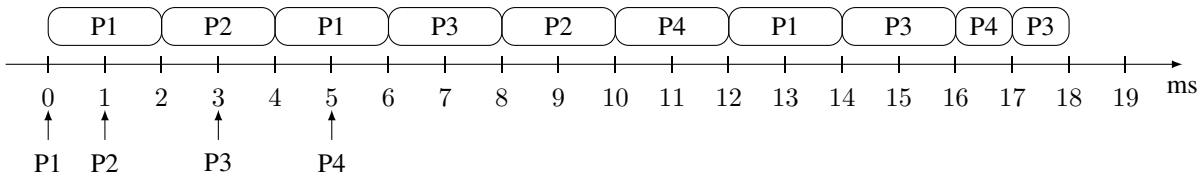


2. ⋆ Le processus P1 est en arrière-plan, donc il n'a rien à faire. Il est prêt.
   ⋆ Le processus P2 attend la carte WIFI. Il est bloqué.
   ⋆ Le processus P3 décode la musique, donc il travaille. Il est élu.
3. Un interblocage survient lorsque plusieurs processus attendent mutuellement une ressource détenue par un autre processus et qu'aucun ne peut continuer.
4. Une situation possible :
   ⋆ P1 obtient MIC ;
   ⋆ P3 obtient CAL ;
   ⋆ P1 attend CAL ;
   ⋆ P3 attend MIC.

   Une situation d'interblocage peut donc se produire.
5. Lorsque plusieurs processeurs sont disponibles sur une machine, plusieurs processus peuvent réellement s'exécuter simultanément. Les performances augmentent et le temps d'attente diminue.
6. ⋆ Avantages : faible consommation énergétique, gain de place
   ⋆ Inconvénients : réparation difficile, évolution matérielle limitée

Partie B

7. On a le chronogramme suivant :



8. Les instructions suivantes conviennent :

```
fp = creer_file_vide()
enfiler(fp, P1)
enfiler(fp, P2)
enfiler(fp, P3)
enfiler(fp, P4)
```

9. Le code suivant convient :

```
def execute_un_processus(file_d_attente, t):
    processus = defiler(file_d_attente)
    if processus['temps'] > quantum:
        processus['temps'] = processus['temps'] - quantum
        enfiler(file_d_attente, processus)
        return t + quantum
    else:
        return t + processus['temps']
```

10. Le code suivant convient :

```
def execute_tous_processus(file_d_attente):
    t = 0
    while not est_vide(file_d_attente):
        t = execute_un_processus(file_d_attente, t)
    return t
```

Exercice 2 (Programmation Python, POO, arbres binaires)
Partie A

1. L'instruction suivante convient : `pegula = Joueuse("Pegula", "Jessica", "USA", 29, 6101)`
2. Le code suivant convient :

```
def ajouter_victoire(self, adversaire):
    self.victoire += 1
    adversaire.defaite += 1
```

3. L'instruction suivante convient : `swiatek.ajouter_victoire(paloni)`
4. Le coût du tri par insertion est quadratique, c'est-à-dire en $O(n^2)$
5. On a le tableau suivant :

Étape	Contenu de `liste_joueuses`
0	`[swiatek, gauff, paloni, sabalenka, pegula]`
1	`[gauff, swiatek, paloni, sabalenka, pegula]`
2	`[gauff, paloni, swiatek, sabalenka, pegula]`
3	`[paloni, gauff, swiatek, sabalenka, pegula]`
4	`[paloni, gauff, swiatek, pegula, sabalenka]`
5	`[paloni, gauff, pegula, swiatek, sabalenka]`

6. Le code suivant convient :

```
def resultat_match(self, score):
    self.score = score
    nb_set_joueuse1 = 0
    nb_set_joueuse2 = 0
    for s1, s2 in score:
        if s1 > s2:
            nb_set_joueuse1 += 1
        else:
            nb_set_joueuse2 += 1
    if nb_set_joueuse1 > nb_set_joueuse2:
        self.gagnante = self.joueuse1
        self.perdante = self.joueuse2
    else:
        self.gagnante = self.joueuse2
        self.perdante = self.joueuse1
    self.gagnante.ajouter_victoire(self.perdante)
```

Partie B

7. Chaque match dépend exactement des deux matchs précédents. Ainsi, chaque nœud a exactement deux fils, ce qui nous donne bien un arbre binaire
8. L'instruction suivante convient :

```
tournoi = Arbre(F,
                Arbre( D1, Arbre(Q1, None, None), Arbre(Q2, None, None)),
                Arbre( D2, Arbre(Q3, None, None), Arbre(Q4, None, None)))
```

9. L'instruction suivante convient : `tournoi.gauche.racine.joueuse1 = gauff`
10. Une fonction récursive est une fonction qui s'appelle elle-même
11. Le code suivant convient :

```
def mise_a_jour(self):
    if self.racine.joueuse1 is None:
        if self.gauche is not None:
            # mise à jour si gagnante à gauche
            if self.gauche.racine.gagnante is not None
                self.racine.joueuse1 = self.gauche.racine.gagnante
            else:
                self.gauche.mise_a_jour()
    if self.racine.joueuse2 is None:
        if self.droit is not None:
            # mise à jour si gagnante à droite
            if self.droit.racine.gagnante is not None
                self.racine.joueuse2 = self.droit.racine.gagnante
            else:
                self.droit.mise_a_jour()
```

Exercice 3 (Bases de données, langage SQL, programmation Python et dictionnaires)

Partie A : bases de données

1. Puisque chaque dossard est unique, cet attribut permet d'identifier un seul coureur. Il peut donc être choisi comme clé primaire
2. Cette requête renvoie les noms et prénoms des coureurs classés par ordre alphabétique du nom
3. La requête suivante convient :

```
SELECT nom, prenom FROM coureur WHERE sexe='F'
```

4. La requête suivante convient :

```
SELECT COUNT(*) FROM coureur
```

5. La requête suivante convient :

```
INSERT INTO coureur VALUES('REMY', 'Patrice', 1973, 'H', 1, 0)
```

6. La requête suivante convient :

```
DELETE FROM coureur WHERE num_dossard=137
```

7. La requête suivante convient :

```
SELECT distance, horaire FROM coureur
JOIN epreuve ON coureur.id_epreuve = epreuve.id_epreuve
WHERE num_dossard = 256
```

8. La requête suivante convient :

```
SELECT num_dossard, nom, prenom, temps FROM coureur
WHERE sexe='F' AND annee < 1986 AND id_epreuve = 2 ORDER BY temps
```

Partie B : programmation Python

9. Cette expression donne la valeur `1000`
10. L'instruction suivante convient : `dict_perf_5km[2026] = [1004,1016,1000,1140,1023,1024]`
11. La fonction suivante convient :

```
def scratch(dico, annee):
    liste = dico[annee]
    meilleur = liste[0]
    for t in liste:
        if t < meilleur:
            meilleur = t
    return meilleur
```

12. La valeur affectée à `o_cat` est `2`
13. On obtient le message d'erreur `local variable 'i_cat'referenced before assignment`
14. L'assertion suivante convient : `assert cat in categories, "Catégorie inconnue"`
15. La fonction `mystere` renvoie la moyenne des temps d'une catégorie. Ici, il s'agit de la catégorie `'SH'`. Cet appel renvoie donc la moyenne $(900 + 1100 + 1000 + 1000 + 1000)/5$, soit `1000`
16. La fonction suivante convient :

```
def records(dico):
    resultats = []
    for categorie in range(6):
        meilleur = 24*3600
        for annee in dico:
            if dico[annee][categorie] < meilleur:
                meilleur = dico[annee][categorie]
        resultats.append(meilleur)
    return resultats
```

Copyright © 2027 FormaV®. Tous droits réservés.

FormaV® est une marque déposée auprès de l'INPI sous le numéro 5227897.

Ce document a été élaboré par FormaV® avec le plus grand soin afin d'accompagner chaque apprenant vers la réussite de ses examens. Son contenu (textes, graphiques, méthodologies, tableaux, exercices, concepts, mises en forme) constitue une œuvre protégée par le droit d'auteur.

Toute copie, partage, reproduction, diffusion ou mise à disposition, même partielle, gratuite ou payante, est strictement interdite sans accord préalable et écrit de FormaV®, conformément aux articles L.111-1 et suivants du Code de la propriété intellectuelle. Dans une logique anti-plagiat, FormaV® se réserve le droit de vérifier toute utilisation illicite, y compris sur les plateformes en ligne ou sites tiers.

En utilisant ce document, vous vous engagez à respecter ces règles et à préserver l'intégrité du travail fourni. La consultation de ce document est strictement personnelle.

Merci de respecter le travail accompli afin de permettre la création continue de ressources pédagogiques fiables et accessibles.