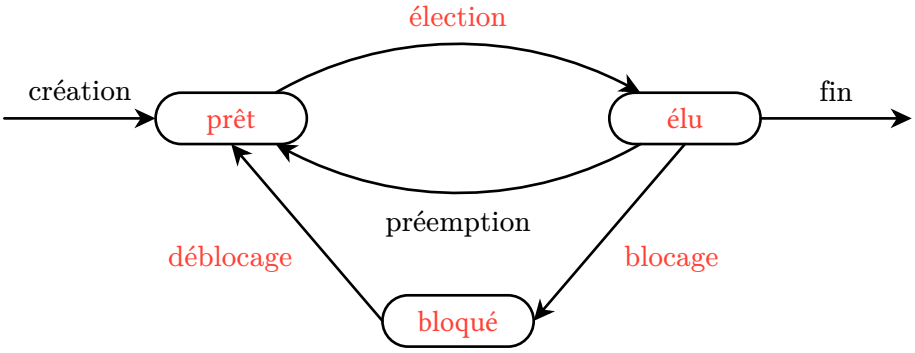
Bac Amérique du Nord Mai 2026, Jour 2

Exercice 1 : Architecture, OS et Structures linéaires (6 points)

Partie A.

1) Schéma classique :



2)
- `P1` est probablement **bloqué** en attente d'un retour de l'application à un état visible où l'interface doit être mise à jour. *Il pourrait également être dans une alternance de élu et prêt avec une faible priorité pour l'ordonnanceur, de façon à garder l'interface à jour pour pouvoir l'afficher instantanément lors d'un retour au premier plan.*
- `P2` est **bloqué** en attente d'un déblocage quand la carte WIFI devient disponible.
- `P3` est **élu**, en cours d'exécution, décodant la musique et l'envoyant au système.

3) Un processus `P1` verrouille l'accès à une ressource `A`, puis `P2` verrouille une ressource `B`, `P1` demande l'accès à `B` (et se retrouve bloqué en attente de la libération de `B` par `P2`), puis finalement `P2` demande l'accès à `A`. `P2` attend donc la libération de `A` par `P1` qui ne se produira jamais puisque `P1` attend `B`... C'est un interblocage.

De façon plus concise, un interblocage se produit lorsque deux processus demandent l'accès exclusif à deux ressources en ordre inverse l'un de l'autre, simultanément, et se retrouvent à attendre mutuellement la libération d'une ressource par l'autre processus.

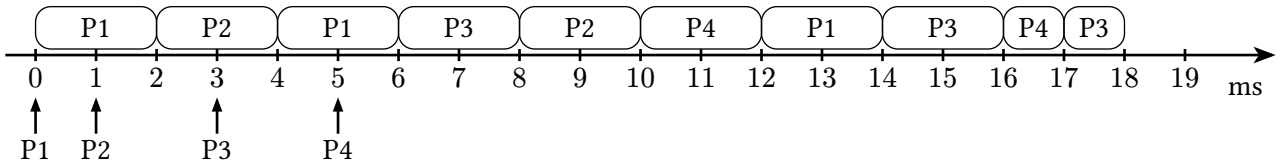
4) `P1` demande `MIC` puis `CAL` tandis que `P2` demande `CAL` puis `MIC` aux mêmes lignes, il peut donc y avoir un interblocage si les premières demandes de `P1` et `P2` sont toutes deux exécutées avant leurs deuxièmes demandes.

5) Disposer de plusieurs processeurs permet d'exécuter plusieurs processus réellement simultanément plutôt que de simuler le multi-processus par une alternance rapide de l'exécution de plusieurs processus sur un seul processeur.

6)
- Avantages des systèmes sur puces :
  - L'intégration (physique et logique) des différents composants permet l'accès plus rapide à la mémoire (ou autre), avec une consommation moindre de courant.
  - La compacité d'un système sur puce permet son emploi dans des appareils plus fins (ou utilisant leur volume à d'autres fins).
- Inconvénients :
  - La réparation et la mise à jour des différents composants est plus difficile que dans un design modulaire, voire impossible.
  - L'évacuation de la chaleur peut également être rendue plus difficile par la compacité du design.

Partie B.

7)



8)

```python
fp = créer_file_vide()
for processus in (P1, P2, P3, P4):
    enfiler(fp, processus)
```

9)

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

10)

```python
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 : POO et Arbres (6 points)

Partie A.

1) `pegula = Joueuse("Pegula", "Jessica", "USA", 29, 6101)`

2)

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

3) `swiatek.ajouter_victoire(paloni)`

4) Dans le pire des cas (*liste strictement décroissante*), le tri par insertion sera **quadratique**, son coût en temps sera proportionnel à n^2. *On dit qu'il est en* $O(n^2)$.

5)

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)

```python
def resultat_match(self, score):
    self.score = score
    nb_set_joueuse1 = 0
    nb_set_joueuse2 = 0
    for (j1, j2) in score:
        if j1 > j2:
            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) Il est possible de modéliser un tournoi de tennis par un arbre binaire car chaque match implique **deux** joueurs, donc chaque match peut avoir pour fils gauche et droit les matchs dont seront issus les joueurs de ce match (les gagnants des fils).

8) En supposant qu'un arbre vide soit représenté par `None` (comme le suggère la fonction `mise_a_jour` à compléter) :

```python
qf1 = Arbre(Q1, None, None)
qf2 = Arbre(Q2, None, None)
qf3 = Arbre(Q3, None, None)
```

```
qf4 = Arbre(Q4, None, None)

df1 = Arbre(D1, qf1, qf2)
df2 = Arbre(D2, qf3, qf4)

tournoi = Arbre(F, df1, df2)
```

ou plus directement :

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

9) `tournoi.racine.joueuse1 = gauff`

10) Une fonction récursive est une fonction qui s'appelle elle-même (dans le corps de la fonction).

11)

```python
def mise_a_jour(self):
    """Met à jour les matchs de l'arbre"""
    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 : BDD, SQL, Python et Dictionnaires (8 points)

Partie A. Bases de données

1) Aucun des autres attributs de la relation `coureur` n'est garanti d'être unique (même les clés composites comme `(nom, prenom)` peuvent théoriquement avoir des doublons). Il est donc nécessaire de créer un attribut dédié pour être la clé primaire, dont la valeur est unique à un enregistrement.

2) Cette requête SQL va renvoyer une relation contenant les attributs `nom` et `prenom` de tous les coureurs, ordonnée par l'ordre alphabétique des noms.

Sur l'extrait donné, elle renverrait donc :

nom	prenom
BODIANE	Lola
BRELET	Sandra
DA SILVA	José
HANG LI	Léo

3)
```sql
SELECT nom, prenom
  FROM coureur
  WHERE sexe = 'F';
```

4)
```sql
SELECT COUNT(*) FROM coureur;
```

5) On n'a pas le numéro de dossard, il faut donc fournir la liste des attributs à affecter et laisser le numéro de dossard être auto-incrémenté :

```sql
INSERT INTO coureur (nom, prenom, sexe, annee, id_epreuve, temps)
  VALUES ('REMY', 'Patrice', 'H', 1973, 1, 0);
```

6)
```sql
DELETE FROM coureur WHERE num_dossard = 137;
```

7)
```sql
SELECT distance, horaire
  FROM coureur c
       JOIN epreuve e ON c.id_epreuve = e.id_epreuve
  WHERE num_dossard = 256;
```

8)
```sql
SELECT num_dossard, nom, prenom, temps
  FROM coureur
  WHERE id_epreuve = 2 AND sexe = 'F' AND annee < 1986
  ORDER BY temps;
```

Il est aussi possible de faire une jointure si l'on ne connaît pas l'identifiant de l'épreuve de 10km :

```sql
SELECT num_dossard, nom, prenom, temps
  FROM coureur c
       JOIN epreuve e ON c.id_epreuve = e.id_epreuve
  WHERE distance = 10 AND sexe = 'F' AND annee < 1986
  ORDER BY temps;
```

Partie B. Programmation Python

9) `dict_perf_5km[2025][2]` est égal à 1000s et correspond au meilleur temps des seniors homme (SH) en 2025.

10) `dict_perf_5km[2026] = [1004, 1016, 1000, 1140, 1023, 1024]`

11)

```python
def scratch(dico, annee):
    temps = dico[annee]
    minimum = temps[0]
    for t in temps:
        if t < minimum:
            minimum = t
    return minimum
```

12) Durant un appel à `mystere(dict_perf_5km, 'SH')`, la valeur 2 sera affectée à `i_cat` (c'est l'indice de "SH" dans `categories`).

13) Le message d'erreur sera **local variable "i_cat" referenced before assignement** car on n'entrera jamais dans le corps du `if cat == categories[i]` où une valeur est affectée à `i_cat`.

14) On pourrait ajouter :
`assert cat in categories, f"{cat} n'est pas une catégorie dans {categories}"`

15) `mystere(dict_perf_5km, 'SH')` nous donnera la moyenne des temps dans la catégorie seniors hommes (SH), le résultat exact dépend de si l'année 2026 a été ajoutée au dictionnaire…

16) On cherche le minimum dans chaque catégorie et puisqu'on suppose qu'aucun temps ne dépasse 24h, on peut initialiser nos temps à `24 * 3600`s (soit 24h en secondes) :

```python
def records(dico):
    mins = [24 * 3600]*6
    for annee in dico:
        temps = dico[annee]
        for i in range(6):
            if temps[i] < mins[i]:
                mins[i] = temps[i]
    return mins
```