BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2026

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 2

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 18 pages numérotées de 1/18 à 18/18.

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

Exercice 1 (6 points)

Cet exercice porte sur la programmation orienté objet, la sécurité des communications.

Une compagnie de taxis est gérée par une application qui a été programmée en langage Python. On s'intéresse à quelques classes programmées dont on donne toutes les propriétés initialisées par le constructeur et les méthodes qui seront utilisées dans cet exercice.

La classe `Chauffeur`

Propriétés (Attributs)

Nom	Description	Type	Initialisation
`nom`	Nom du chauffeur.	`str`	Premier paramètre passé au constructeur.
`prenom`	Prénom du chauffeur.	`str`	Deuxième paramètre passé au constructeur.
`neph`	Numéro d'enregistrement préfectoral harmonisé, composé de 12 chiffres. Numéro unique attribué à vie depuis 2013.	`str`	Troisième paramètre passé au constructeur.

La classe `Client`

Propriétés (Attributs)

Nom	Description	Type	Initialisation
`nom`	Nom du client.	`str`	Premier paramètre passé au constructeur.
`prenom`	Prénom du client.	`str`	Deuxième paramètre passé au constructeur.
`adresse`	Adresse du client	`str`	Troisième paramètre passé au constructeur.
`nb_personne`	Nombre de personnes accompagnant le client.	`int`	Quatrième paramètre passé au constructeur.

La classe `Taxi`

Propriétés (Attributs)

Nom	Description	Type	Initialisation
`immatriculation`	Immatriculation du véhicule.	`str`	Premier paramètre passé au constructeur.
`type_vehicule`	Le type du véhicule : `'standard', 'monospace', 'minibus'`	`str`	Deuxième paramètre passé au constructeur.
`energie`	Hybride, électrique ou thermique.	`str`	Troisième paramètre passé au constructeur.
`adaptation`	Adapté pour les personnes à mobilité réduite.	`bool`	Quatrième paramètre passé au constructeur.
`libre`	La voiture peut-elle prendre des clients?	`bool`	Initialisé à `True` dans le constructeur.
`chauffeur`	Le chauffeur de la compagnie qui conduit le véhicule.	`Chauffeur`	Initialisé à `None` dans le constructeur.

Méthodes

Nom	Description	Paramètres	Type de renvoie
`est_libre`	Renvoie la valeur de la propriété `libre`.	Aucun	`bool`
`modifier_libre`	Modifie la propriété `libre`.	`statut` de type `bool`	Aucun
`choix_chauffeur`	Met à jour la propriété `chauffeur`.	`chauffeur` (de type `Chauffeur`)	Aucun

La classe `Course`

Propriétés (Attributs)

Nom	Description	Type	Initialisation
`client`	Le client qui a commandé la course.	`Client`	Premier paramètre passé au constructeur.
`taxi`	Le véhicule qui prendra en charge le client et les accompagnants.	`Taxi`	Deuxième paramètre passé au constructeur.
`depart`	Adresse de départ au format : `'42 rue de l'informatique, Octet-sur-Mer, France'`.	`str`	Troisième paramètre passé au constructeur.
`destination`	Adresse de destination au format : `'42 rue de l'Informatique, Octet-sur-Mer, France'`.	`str`	Quatrième paramètre passé au constructeur.
`date_depart`	Date de départ au format : `datetime(année, mois, jour, heure, minutes)`.	`datetime`	Initialisée à `None` dans le constructeur.
`date_arrivee`	Date d'arrivée au format : `datetime(année, mois, jour, heure, minutes)`.	`datetime`	Initialisée à `None` dans le constructeur.

Méthodes

Nom	Description	Paramètres	Type de renvoie
`distance`	Renvoie la distance du trajet en km.	Aucun	`float`
`temps`	Renvoie le temps du trajet en minutes.	Aucun	`float`

On dispose des variables suivantes.

- Un dictionnaire `vehicules` qui a pour clés le type du véhicule et pour valeurs un dictionnaire regroupant les informations suivantes.
  - `capacite`: le nombre de passagers pris en charge en plus du conducteur.
  - `prix_km` : le prix du km appliqué par la compagnie en euros.
  - `prise_en_charge` : le coup fixe de la prise en charge en euros.
  - `tarif_horaire` : le tarif horaire appliqué en euros.

```
vehicules = {
    'standard': {'capacite': 4, 'prix_km': 1.10,
'prise_en_charge': 3, 'tarif_horaire': 38},
    'monospace': {'capacite': 8, 'prix_km': 1.80,
'prise_en_charge': 9, 'tarif_horaire': 60},
    'minibus': {'capacite': 19, 'prix_km': 3.00,
'prise_en_charge': 50, 'tarif_horaire': 100}
    }
```

- Une liste `energie` contenant les éléments d'information sur l'énergie utilisée par les véhicules de la flotte de taxis de la compagnie.

```
energie = ['hybride', 'électrique', 'thermique']
```

Partie A

Chaque classe est définie dans un script Python qui porte le nom de la classe, `taxi.py` pour la classe `Taxi`, `client.py` pour la classe `Client`, etc., et les variables `vehicules` et `energie` sont déclarées dans un script Python nommé `donnees.py`. L'application est programmée dans un script Python nommé `application.py`.

Tous les fichiers sont dans le même répertoire.

1. Donner les instructions à ajouter au script `application.py` pour que l'on puisse y utiliser les variables `vehicules` et `energie` du script `donnees.py` et les classes `Chauffeur`, `Taxi`, `Client` et `Course`.

2. Écrire une commande Python permettant d'instancier le chauffeur de taxi de prénom John et de nom Doe, dont le numéro neph est 140159320012.

Le chauffeur John Doe conduit le véhicule de type standard à énergie électrique d'immatriculation HG-818-AV, qui n'est pas adapté pour les personnes à mobilités réduites.

3. Écrire les instructions qui permettent de créer le véhicule d'immatriculation HG-818-AV et de l'attribuer au chauffeur John Doe.

La centrale de la compagnie de taxi envoie une demande au chauffeur John Doe pour qu'il prenne en charge la cliente Jeanne Doe qui est accompagnée de ses deux enfants.

Elle désire, de son domicile au 6 rue des ordinateurs à Paris, se rendre au 211 avenue Jean Jaurès à Paris (Le parc de la Villette).

4. Écrire une assertion, qui placée au début du constructeur de la classe `Course`, permet d'interdire la création de la course si le taxi n'est pas libre.

5. Ajouter une instruction à la classe `Course` afin que le taxi ne soit plus marqué comme libre lors de la création d'une course.

6. Écrire la ou les instructions qui permettent de créer la course de la cliente Jeanne Doe.

À l'aide de la bibliothèque Python `datetime` on peut calculer la différence de temps en secondes entre deux dates à l'aide de la fonction du même nom `datetime`.

```
>>> from datetime import datetime
>>> depart = datetime(2025, 7, 12, 9, 12)
>>> arrivee = datetime(2025, 7, 12, 17, 12)
>>> (arrivee - depart).total_seconds()
28800.0
```

C'est-à-dire qu'entre le 12 juillet 2025 à 9h12 et le 12 juillet 2025 à 17h12 il s'est écoulé 28800 secondes.

7. Écrire un code Python pour la méthode `temps` de la classe `Course` qui renvoie le temps du trajet en minutes.

On veut ajouter la méthode `tarif` à la classe `Course` qui permet de renvoyer le montant à payer en euros par le client pour la course réalisée. Le chauffeur John Doe va réaliser un trajet de 9,1 km en 19 minutes pour amener sa cliente Jeanne Doe à destination. Pour un véhicule standard la prise en charge est de 3 euros, le prix au km est de 1,10 euros et le tarif horaire est de 38 euros.

Le calcul du coût en euros de la course est le suivant :

$$3 + 1{,}10 \times 9{,}1 + 38 \times \frac{19}{60} \approx 25{,}04$$

8. Recopier et compléter le code Python pour la méthode `tarif` de la classe `Course` suivant.

```
 1     def tarif(self):
 2         type_vehicule = ...
 3         donnees_tarifaire = vehicules[type_vehicule]
 4         prise_en_charge = ...
 5         tarif_h = ...
 6         prix_km = ...
 7         tarif = ...
 8         tarif = tarif + prix_km*...
 9         tarif = tarif + tarif_h*.../60
10         return tarif
```

Partie B

L'application permet de gérer les échanges entre la centrale de la compagnie de taxi et les différents chauffeurs. Le système permet de communiquer de façon sécurisée avec les chauffeurs.

9. Expliquer la différence de fonctionnement entre les chiffrements symétrique et asymétrique.

Pour initier une communication avec le chauffeur John Doe, l'application va générer aléatoirement une clé de session symétrique avec laquelle elle va chiffrer les messages à transmettre. Le chauffeur doit posséder cette clé pour déchiffrer les messages. L'application transmet la clé au chauffeur sans traitement particulier, avant d'envoyer les messages chiffrés.

10. Expliquer le problème de sécurité qui peut intervenir en transmettant la clé de session directement au chauffeur.

Il est établi que le chiffrement symétrique est très rapide et efficace pour chiffrer de grandes quantités de données. La compagnie de taxi souhaite que les communications soient sécurisées, sans pour autant ralentir la transmission des échanges entre la centrale et les chauffeurs.

11. Proposer une stratégie qui permettrait de transmettre cette clé de session symétrique de façon sécurisée en détaillant les étapes.

Exercice 2 (6 points)

Cet exercice porte sur les bases de données relationnelles, les requêtes SQL, les réseaux et les protocoles de routage.

Dans cet exercice, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT`, `FROM`, `WHERE` (avec les opérateurs logiques `AND`, `OR`), `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT`, `DELETE` ;
- affiner les recherches à l'aide de `DISTINCT`, `ORDER BY`.

On rappelle aussi que la fonction `COUNT` permet de compter le nombre d'enregistrements extraits lors d'une requête. Par exemple, la requête :

```
SELECT COUNT(*) FROM ordinateur;
```

permet d'afficher le nombre d'ordinateurs présents dans la relation `ordinateur`.

Dans un schéma relationnel, on utilisera les conventions suivantes :

- la clé primaire d'une relation est soulignée ;
- les attributs précédés de # sont les clés étrangères.

Ada est une passionnée d'informatique vintage. Elle collectionne de vieux ordinateurs des années 80 et 90, qu'elle restaure avec soin.

Partie A

Pour gérer sa collection d'ordinateurs, Ada a conçu une petite base de données relationnelle qui contient la relation `ordinateur` dont le schéma relationnel, avec les domaines, est le suivant :

`ordinateur (id_ordi:INT, marque:TEXT, modele:TEXT, etat:TEXT)`

Voici un extrait de la relation `ordinateur`.

ordinateur			
id_ordi	marque	modele	etat
5	Atari	1040 ST	Fonctionnel
6	Commodore	Commodore 64	Réparation
7	Sinclair	ZX Spectrum	Fonctionnel

ordinateur			
8	Commodore	Amiga 500	Fonctionnel
9	Apple	Apple IIe	Réparation
10	Commodore	Vic 20	Fonctionnel
11	Sinclair	ZX 81	Panne
12	Atari	800 XL	Fonctionnel

1. Écrire une requête SQL permettant d'afficher tous les modèles des ordinateurs en réparation présents dans la collection.

Ada vient d'acquérir un nouvel ordinateur, un modèle MO5 de la marque Thomson en panne.

2. Écrire une requête SQL permettant d'insérer ce nouvel ordinateur dans la base de données avec un `id_ordi` de 22.

Ada a restauré l'Apple IIe (`id_ordi` = 9) qui est maintenant fonctionnel.

3. Écrire une requête SQL permettant de mettre à jour son état dans la base de données.

4. Donner le résultat de la requête suivante appliquée à l'extrait de table donné précédemment.

```
SELECT COUNT(*) FROM ordinateur
WHERE marque = 'Commodore' AND etat = 'Fonctionnel';
```

Partie B

Ada souhaite maintenant intégrer sa collection de jeux vidéos vintages dans la base de données. Pour cela elle y ajoute une relation `jeu` (de clé primaire `id_jeu`) et une relation `plateforme` (de clé primaire `id_plat`). Elle ajoute aussi un attribut `id_plat` dans la relation `ordinateur`.

Voici un extrait des trois relations `ordinateur`, `jeu` et `plateforme`.

ordinateur				
id_ordi	marque	modele	etat	id_plat
5	Atari	1040 ST	Fonctionnel	2
6	Commodore	Commodore 64	Réparation	1

ordinateur				
7	Sinclair	ZX Spectrum	Fonctionnel	5
8	Commodore	Amiga 500	Fonctionnel	3
9	Apple	Apple IIe	Fonctionnel	4
10	Commodore	Vic 20	Fonctionnel	6
11	Sinclair	ZX 81	Panne	8
12	Atari	800 XL	Fonctionnel	9

jeu				
id_jeu	titre	genre	etat	id_plat
12	Lemmings	Puzzle	Complet	3
13	Barbarian	Action	Sans notice	1
14	Populous	Stratégie	Complet	2
15	Knight Lore	Aventure	Loose	5
16	Dungeon Master	RPG	Complet	2

plateforme		
id_plat	nom	bits
1	C64	8
2	Atari ST	16
3	Amiga	16
4	Apple II	8
5	Spectrum	8

Dans les relations `ordinateur` et `jeu`, l'attribut `id_plat` est une clé étrangère qui fait référence à la clé primaire `id_plat` de la relation `plateforme`.

5. Donner le schéma relationnel, avec les domaines, des relations `jeu` et `plateforme`.

6. Écrire une requête SQL permettant d'afficher tous les titres des jeux compatibles avec la plateforme de nom Oric présents dans la collection.

7. Écrire une requête SQL permettant d'afficher tous les titres des jeux compatibles avec le modèle d'ordinateur Amstrad 6128 présents dans la collection.

Partie C

Avec des amis, Ada a créé une plateforme appelée VintagePixel qui héberge plusieurs serveurs dédiés à des jeux rétros en ligne. Chaque serveur, situé sur un réseau local, est dédié à un type de jeu (serveur A pour les jeux d'arcade, serveur S pour les jeux de stratégie, serveur P pour les jeux de plateforme et serveur C pour les jeux de course). Les serveurs sont interconnectés via des routeurs pour permettre aux joueurs de s'y connecter et de jouer ensemble. Le schéma du réseau est donné ci-dessous.

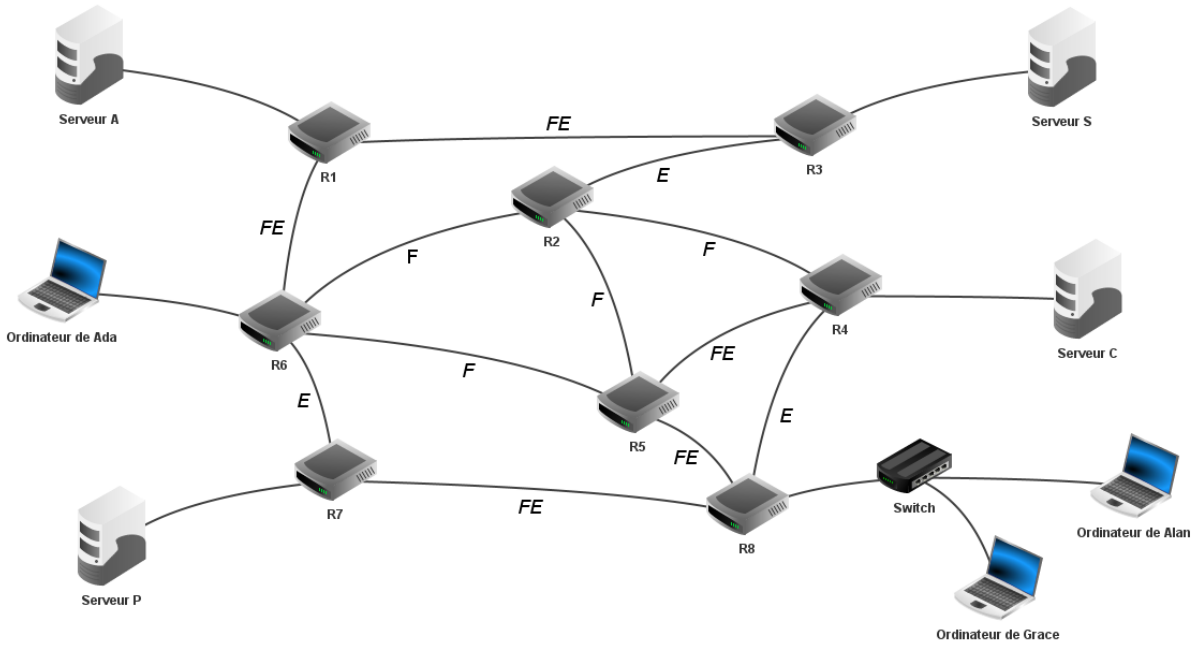


Figure 1. Schéma du réseau

Les codes indiqués sur les liaisons entre routeurs correspondent aux types de liaisons :

- F pour une liaison Fibre de bande passante 1 Gbit/s ;
- FE pour une liaison Fast Ethernet de bande passante 100 MBits/s ;
- E pour une liaison Ethernet de bande passante 10 MBits/s.

Alan et Grace, utilisateurs de la plateforme VintagePixel, sont connectés sur un même réseau local. L'adresse IP de l'ordinateur de Grace est 192.168.208.11 et le masque de sous-réseau du réseau local est 255.255.255.240.

L’ordinateur d’Alan ne parvient pas à se connecter au réseau local.

8. Donner la commande qu’Alan doit utiliser dans le terminal pour tester la connectivité entre son ordinateur et l’ordinateur de Grace.

Une fois le problème de connexion d’Alan réglé, Alan et Grace souhaitent inviter des amis à se connecter sur ce réseau local.

9. Convertir le nombre 240 en binaire puis déterminer le nombre de machines qu’il est encore possible d’ajouter à ce réseau local.

Dans un premier temps, on s’intéresse au protocole RIP, qui minimise le nombre de sauts entre routeurs. Le début de la table de routage simplifiée du routeur R8 est donnée ci-dessous.

Routeur R8		
Destination	Routeur suivant	nombre de sauts
R1	….	….
R2	….	….
R3	….	….
R4	….	….
R5	R5	1
R6	R7	2
R7	R7	1

10. Recopier et compléter les quatre premières lignes de la table du routage simplifiée du routeur R8.

Alan veut envoyer un message à Ada.

11. Donner un chemin pouvant être suivi par un paquet de données.

On s’intéresse maintenant au protocole de routage OSPF. Ce dernier cherche à minimiser la somme des coûts des liaisons entre les routeurs empruntés par un paquet. Le coût des liaisons est ici de 1 pour une liaison fibre (F), de 10 pour une liaison Fast-Ethernet (FE) et de 100 pour une liaison Ethernet (E).

Grace veut se connecter au serveur S pour jouer à son jeu de stratégie préféré.

12. Déterminer le chemin suivi par un paquet de données pour aller du routeur R8 au routeur R3 et donner son coût en respectant le protocole OSPF.

Exercice 3 (8 points)

Cet exercice porte sur l'architecture matérielle (dont le langage assembleur), les structures linéaires de données et la programmation orientée objet.

Les deux parties peuvent être traitées indépendamment.

Partie A

L'architecture moderne des ordinateurs est basée sur un modèle conçu en 1945 dans lequel la mémoire vive de l'ordinateur (la RAM) est utilisée pour stocker à la fois les instructions et les données demandées ou produites par les programmes.

1. Donner le nom de l'inventeur de ce modèle, ayant donné son nom au modèle.
2. Sur la copie, associer un des termes parmi les propositions suivantes à chaque numéro sur le schéma ci-dessous représentant cette architecture : 'Mémoire RAM', 'Processeur', 'Unité arithmétique et logique' et 'Unité de contrôle'.

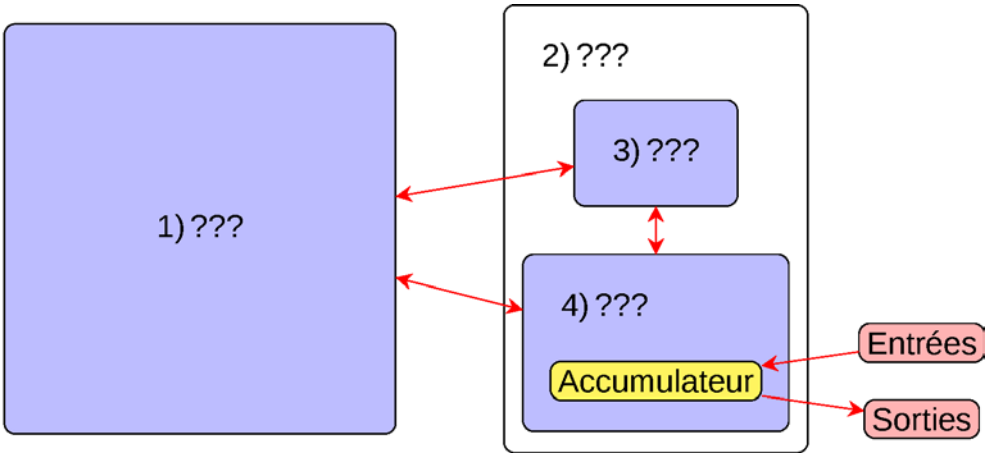


Figure 1. Modèle d'architecture moderne des ordinateurs.

3. Expliquer ce que signifie le fait que la mémoire RAM soit 'volatile'.
4. Classer les types de mémoire suivants selon leur rapidité d'accès aux données, de la plus rapide à la plus lente.
   - Mémoire cache
   - Disque dur
   - Mémoire vive
   - Registre

Les instructions d'un programme Python ne peuvent pas être exécutées telles quelles par l'unité de contrôle, c'est pourquoi elles sont transformées en langage machine. Ainsi, un programme écrit en Python qui contient une centaine de lignes peut correspondre à quelques dizaines de milliers d'octets en *langage machine* à faire exécuter par l'unité de contrôle.

Pour pallier la difficulté d'écrire de très grandes suites d'octets, des langages très bas niveau (c'est-à-dire proches du langage machine) mais lisibles par les humains ont été conçus. C'est le cas du langage assembleur RISC-V sur 32 bits qui est étudié ici.

Dans un code écrit en assembleur, chaque ligne est soit une instruction qui permet de manipuler des variables appelées *registres*, soit une étiquette qui permet d'identifier une position dans le code. Par exemple, le code ci-dessous permet de mettre dans le registre `r5` l'addition entre la valeur du registre `r2` et la valeur `3` si la valeur du registre `r7` est strictement supérieure à la valeur du registre `r6`, puis ajoute `1` au registre `r5` dans tous les cas. (On suppose que l'unité de contrôle dispose de 8 registres notés `r1` à `r8`).

```
    ble r7, r6, SUITE
    addi r5, r2, 3
SUITE:
    addi r5, r5, 1
```

Voici un extrait des instructions disponibles en RISC-V sur 32 bits :

- `addi rA, rB, v` : permet de placer dans le registre `rA` la somme de la valeur du registre `rB` et de la valeur `v` ;
- `add rA, rB, rC` : permet de placer dans le registre `rA` la somme de la valeur du registre `rB` et de celle du registre `rC` ;
- `sub rA, rB, rC` : permet de placer dans le registre `rA` la valeur du registre `rB` moins celle du registre `rC` ;
- `beq rA, rB, label` : saute vers l'instruction après l'étiquette `label` si la valeur du registre `rA` est égale à la valeur du registre `rB` ;
- `bne rA, rB, label` : saute vers l'instruction après l'étiquette `label` si la valeur du registre `rA` n'est pas égale à la valeur du registre `rB` ;
- `bgt rA, rB, label` : saute vers l'instruction après l'étiquette `label` si la valeur du registre `rA` est strictement plus grande que la valeur du registre `rB` ;
- `ble rA, rB, label` : saute vers l'instruction après l'étiquette `label` si la valeur du registre `rA` est plus petite ou égale à la valeur du registre `rB`.

5. Écrire une instruction en assembleur permettant de mettre dans le registre `r2` le résultat de la soustraction entre la valeur du registre `r5` et celle du registre `r4`.

6. Expliquer ce que réalise l'instruction `add r1 r2 0`.

7. Écrire une suite d'instructions en assembleur permettant d'échanger les contenus des registres `r1` et `r2`. On pourra utiliser le registre `r3` pour stocker temporairement des valeurs.

Au niveau matériel, les opérations sont effectuées dans l'unité arithmétique et logique qui contient des circuits logiques booléens combinatoires. On donne ci-dessous les portes logiques effectuant les opérations booléennes classiques.

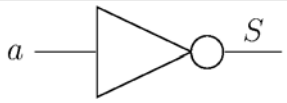
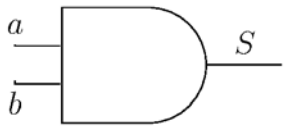
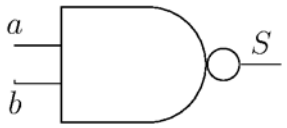
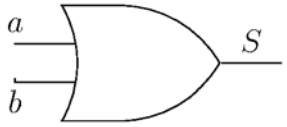
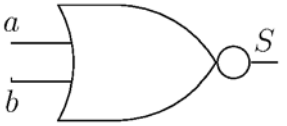
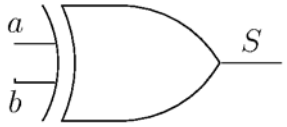
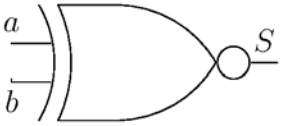
Nom	Symbole	Table de vérité
NOT		a | S 0 | 1 1 | 0
AND		a | b | S 0 | 0 | 0 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1
NAND		a | b | S 0 | 0 | 1 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0
OR		a | b | S 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 1
NOR		a | b | S 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 0
XOR		a | b | S 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0
NXOR		a | b | S 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1

Figure 2. Portes logiques.

Dans une table de vérité, a et b représentent chacun une valeur binaire d'entrée de la porte et S représente la valeur binaire de sortie de la porte.

8. Donner pour chaque lettre A, B, C et D sa valeur binaire dans la table de vérité du circuit combinatoire ci-après.

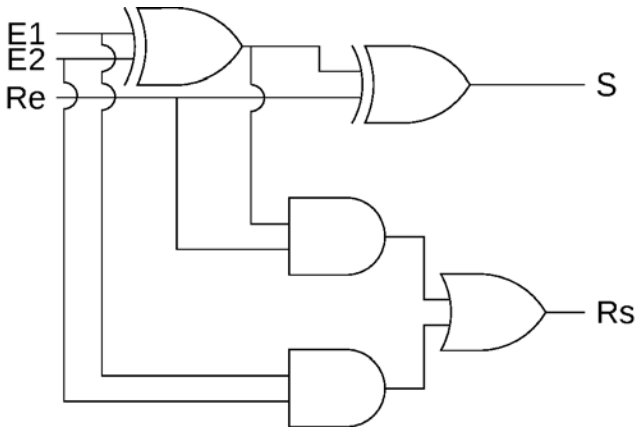


Figure 3. Circuit logique

Table de vérité				
E1	E2	Re	**Rs**	**S**
0	0	0	0	0
0	0	1	**A**	**B**
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	**C**	0
1	1	0	1	0
1	1	1	1	**D**

9. En observant le tableau complété à la question précédente, expliquer ce que fait le circuit étudié.

Partie B

Une structure souvent utilisée dans les mémoires des systèmes modernes est la *mémoire tampon*. Il s'agit d'une zone de la mémoire réelle utilisée pour entreposer temporairement des données, par exemple par deux processus ne travaillant pas au même rythme et ayant pourtant besoin d'être synchronisés.

Cette mémoire tampon fonctionne comme une file, ainsi les données sont stockées dans l'ordre chronologique de leur enregistrement et sont sorties de la mémoire tampon dans le même ordre.

10. Entre FIFO et LIFO, donner le terme correspondant le mieux à la structure de file.

La file représentant la mémoire tampon est un tableau circulaire, c'est-à-dire que lorsque l'on souhaite ajouter un élément après la dernière case du tableau, on le place dans la première case du tableau.

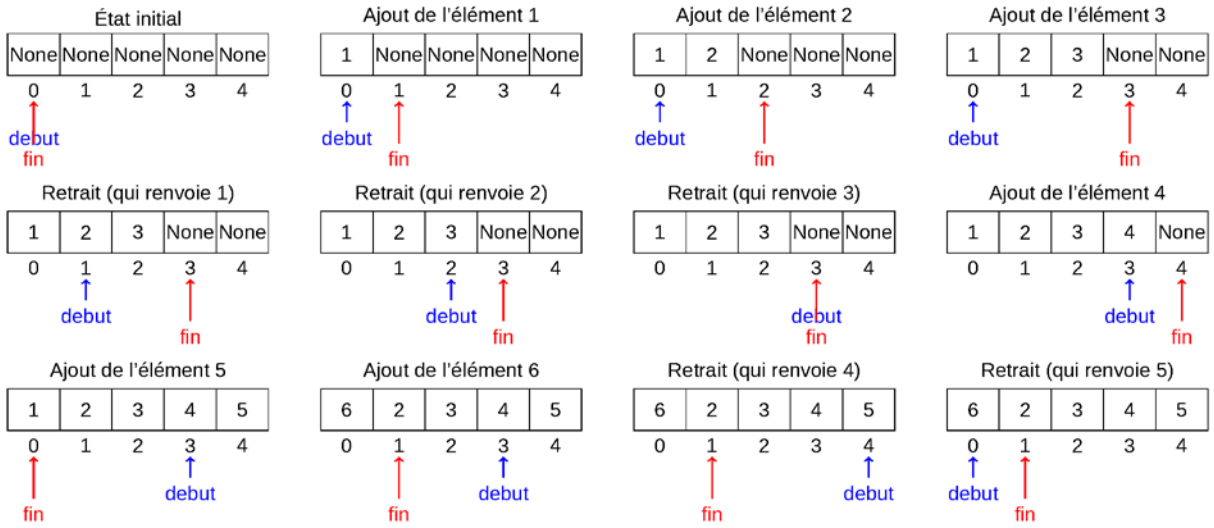


Figure 4. Exemple de manipulation d'une mémoire tampon de capacité 5.

On remarque que le retrait d'un élément de la mémoire tampon consiste simplement au décalage de l'indice de début, sans retirer réellement l'élément du tableau. On suppose que la capacité maximale d'une file n'est jamais atteinte, c'est-à-dire que les indices de début et de fin ne sont égaux que lorsque la file est vide.

La classe `Tampon` ci-dessous implémente une mémoire tampon à l'aide d'une liste Python. La longueur de cette liste est fixée définitivement à l'initialisation à l'aide du paramètre `capacite` du constructeur.

```
 1 class Tampon:
 2     def __init__(self, capacite):
 3         self.capacite = capacite
 4         self.contenu = [None for i in range(capacite)]
 5         self.debut = 0 # Indice de la tête de file
 6         self.fin = 0   # Indice de la queue de file exclu
 7
 8     def ajouter_element(self, element):
 9         self.contenu[self.fin] = element
10         self.fin = self.fin + 1
11         if self.fin == self.capacite:
12             self.fin = 0
13
14     def retirer_element(self):
15         ...
16
```

```
17      def est_vide(self):
18          return ...
```

11. Donner la valeur de l'attribut `contenu` de l'objet `memoire` à l'issue de l'appel suivant :

    ```
    memoire = Tampon(5)
    ```

On considère avoir à disposition une mémoire tampon dont l'état est représenté ci-dessous.

Figure 5. État de la mémoire tampon à considérer pour la question suivante.

12. Donner l'état de cette mémoire tampon, avec l'éventuel élément renvoyé, après chacune des opérations successives suivantes :

    a) retrait d'un élément ;
    b) ajout de l'élément 10 ;
    c) retrait d'un élément ;
    d) retrait d'un élément ;
    e) ajout de l'élément 3 ;
    f) retrait d'un élément ;
    g) retrait d'un élément.

13. Donner la valeur de l'attribut `contenu` de la mémoire tampon `memoire` après l'exécution du programme suivant.

    ```
    memoire = Tampon(5)
    memoire.ajouter_element('n')
    memoire.ajouter_element('s')
    memoire.ajouter_element('i')
    memoire.retirer_element()
    memoire.ajouter_element('n')
    memoire.ajouter_element('f')
    memoire.retirer_element()
    memoire.ajouter_element('o')
    ```

14. Recopier et compléter la méthode `est_vide` de la classe `Tampon`.

15. Écrire le code de la méthode `retirer_element` de la classe Tampon qui renvoie l'élément en tête de file et le retire de la file. On suppose que la file est non-vide. On rappelle que l'élément n'est pas réellement retiré de la liste, seul l'indice de début est décalé.