Math93.com

BAC NSI - Correction
Amérique du Nord - 2021 - NSI

Thèmes des exercices

Le candidat doit choisir **3 exercices** qu'il traitera sur les 5 exercices proposés.

- Exercice 1 : données relationnelles et le langage SQL.
- Exercice 2 : les notions de routage, de processus et de systèmes sur puces.
- Exercice 3 : les tableaux et sur la programmation de base en Python.
- Exercice 4 : les arbres binaires et leurs algorithmes associés.
- Exercice 5 : la notion de pile, de file et sur la programmation de base en Python.

Exercice 1. Données relationnelles et le langage SQL — 4 points

On donne ci dessous le schéma relationnel de la relation Ordinateur, suivi d'un extrait de la relation Ordinateur. La clé primaire est soulignée.
Ordinateur(nom_ordi : String, salle : String, marque_ordi : String, modele_ordi : String, annee : Int, video : Boolean)
Les 5 premières lignes de la relation Ordinateur :

nom_ordi	salle	marque_ordi	modele_ordi	annee	video
Gen-24	012	HP	compaq pro 6300	2012	true
Tech-62	114	Lenovo	p300	2015	true
Gen-132	223	Dell	Inspiron Compact	2019	true
Gen-133	223	Dell	Inspiron Compact	2019	false
Gen-134	223	Dell	Inspiron Compact	2019	false

1.

(a) À l'aide d'un système de gestion de base de données, on envoie au serveur la requête SQL suivante :

```
SELECT salle, marque_ordi FROM Ordinateur ;
```

Quel résultat produit cette requête sur l'extrait de la relation Ordinateur donné ci-dessus ?

Corrigé

On obtient la liste des marques de chaque ordinateur présent dans chaque salle.

Les 5 premières lignes seront :

salle	marque_ordi
012	HP
114	Lenovo
223	Dell
223	Dell
223	Dell

(b) Quel résultat produit la requête suivante sur l'extrait de la relation Ordinateur donné ci-dessus ?

```
SELECT nom_ordi, salle FROM Ordinateur WHERE video = true ;
```

Corrigé

On obtient la liste des ordinateurs et leur salle associée qui sont connectés à un vidéoprojecteur.

Les 3 premières lignes seront :

nom_ordi	salle
Gen-24	012
Tech-62	114
Gen-132	223

2. Écrire une requête SQL donnant tous les attributs des ordinateurs correspondant aux années supérieures ou égales à 2017 ordonnées par dates croissantes.

Corrigé

Il faut taper la requête SQL ci-dessous :

```
SELECT * FROM ordinateur WHERE annee>=2017 ORDER BY annee ;
```

3.

(a) Pour quelle raison l'attribut salle ne peut-il pas être une clé primaire pour la relation Ordinateur ?

Corrigé

Dans une base de données relationnelle, une **clé primaire** est la donnée qui permet d'identifier de manière unique un enregistrement (une ligne) dans une table.

De ce fait, l'attribut salle ne peut pas être une clé primaire pour la relation Ordinateur puisqu'une salle peut contenir plusieurs ordinateurs.

Le nom de la salle n'est pas associé à un unique ordinateur, par exemple la salle 223 en contient 3.

(b) On donne ci-dessous un extrait de la relation Imprimante :
Les 5 premières lignes de la relation Imprimante

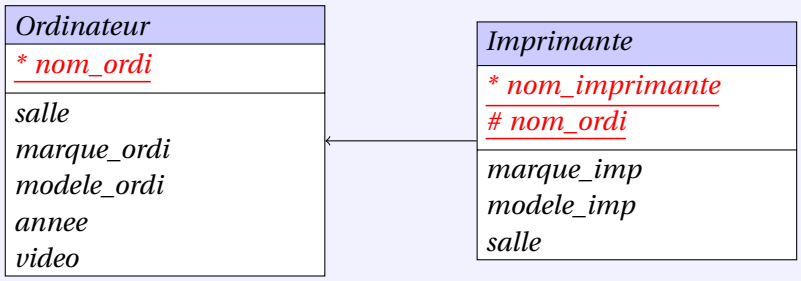
nom_imprimante	marque_imp	modele_imp	salle	nom_ordi
imp_BTS_NB	HP	Laserjet pro M15w	114	Tech-62
imp_BTS_Couleur	Canon	Megatank Pixma G5050	114	Tech-62
imp_salle-info1	Brother	2360DN	223	Gen-132
imp_salle-info1	Brother	2360DN	223	Gen-133
imp_salle-info1	Brother	2360DN	223	Gen-134

On prend (nom_imprimante, nom_ordi) comme clé primaire.

Écrire le schéma relationnel de la relation Imprimante en précisant les éventuelles clés étrangères pour les autres relations.

Corrigé

Voici le schéma relationnel de la relation Imprimante avec les clés étrangères pour les autres relations.



* : clé primaire

\# : clé étrangère

La clé primaire est soulignée dans chaque table.

4. On donne ci-dessous un extrait de la relation Videoprojecteur :

   Les 4 premières lignes de la relation Videoprojecteur

salle	marque_video	modele_video	tni
012	Epson	xb27	true
114	Sanyo	PLV-Z3	false
223	Optoma	HD143X	false
225	Optoma	HD143X	true

   (a) Écrire une requête SQL pour ajouter à la relation Videoprojecteur le vidéoprojecteur nouvellement installé en salle 315 de marque NEC, modèle ME402X et non relié à un TNI.

Corrigé

Il faut écrire la requête suivante :

```
INSERT INTO 'Videoprojecteur'
('salle','marque_video','modele_video','tni')
VALUES
('315','NEC','ME402X',false);
```

   (b) Écrire une requête SQL permettant de récupérer les attributs salle, nom_ordi, marque_video des ordinateurs connectés à un vidéoprojecteur équipé d'un TNI.

Corrigé

On peut écrire les requêtes suivantes :

```
SELECT o.salle, nom_ordi, marque_video
FROM Ordinateur AS o, Videoprojecteur As v
WHERE o.salle=v.salle
AND o.video = true
```

ou encore avec une jointure

```
SELECT o.salle, nom_ordi, marque_video
FROM Ordinateur AS o
JOIN Videoprojecteur As v
ON o.salle=v.salle
WHERE o.video = true
```

Exercice 2. Les notions de routage, de processus et de systèmes sur puces — 4 points

Un constructeur automobile utilise des ordinateurs pour la conception de ses véhicules. Ceux-ci sont munis d'un système d'exploitation ainsi que de nombreuses applications parmi lesquelles on peut citer :

— un logiciel de traitement de texte ;

— un tableur ;

— un logiciel de Conception Assistée par Ordinateur (CAO) ;

— un système de gestion de base de donnée (SGBD).

Chaque ordinateur est équipé des périphériques classiques : clavier, souris, écran et est relié à une imprimante réseau.

1. Ce constructeur automobile intègre à ses véhicules des systèmes embarqués, comme par exemple un système de guidage par satellites (GPS), un système de freinage antiblocage (ABS) ... Ces dispositifs utilisent des systèmes sur puces (SoC : Système on a Chip).

   Citer deux avantages à utiliser ces systèmes sur puces plutôt qu'une architecture classique d'ordinateur.

Corrigé

Un SoC, *System on a Chip*, accueille sur une même puce un microprocesseur (CPU), de la mémoire vive (RAM) un circuit graphique (GPU) et des composants Wifi, Bluetooth, etc. et tout cela dans une taille très réduite, ce qui est idéal pour des dispositifs mobiles.

On peut citer plusieurs avantages à utiliser ces systèmes sur puces plutôt qu'une architecture classique d'ordinateur.

— Avantages.
  — Faible encombrement : cette miniaturisation permet de gagner en taille et poids.
  — Cette miniaturisation permet aussi d'éviter d'intégrer des système de refroidissement lourds et volumineux.
  — Faible consommation d'énergie.
  — Faibles coûts de fabrication.

— Inconvénients.
  — Par contre si un des composants est défaillant, il faut remplacer tout le SoC.
  — La durée de vie et la fiabilité d'un SoC sont celles de son maillon le plus faible, le premier élément à présenter une défaillance provoquera une altération du SoC.

2. Un ingénieur travaille sur son ordinateur et utilise les quatre applications citées au début de l'énoncé. Pendant l'exécution de ces applications, des processus mobilisent des données et sont en attente d'autres données mobilisées par d'autres processus. On donne ci-dessous un tableau indiquant à un instant précis l'état des processus en cours d'exécution et dans lequel D1, D2, D3, D4 et D5 sont des données. La lettre M signifie que la donnée est mobilisée par l'application; la lettre A signifie que l'application est en attente de cette donnée. Lecture du tableau : le logiciel de traitement de texte mobilise (M) la donnée D1 et est en attente (A) de la donnée D2.

	D1	D2	D3	D4	D5
Traitement de texte	M	A	-	-	-
Tableur	A	-	-	-	M
SGBD	-	M	A	A	-
CAO	-	-	A	M	A

Montrer que les applications s'attendent mutuellement. Comment s'appelle cette situation?

Corrigé

Les applications s'attendent mutuellement car des données sont mobilisée en même temps par les applications.

- Le Traitement de texte est en attente de D2 mobilisée par SGBD
- SGBD est en attente de D4 mobilisée par CAO
- CAO est en attente de D5 mobilisée par le Tableur
- Le Tableur est en attente de D1 mobilisée par le Traitement de texte qui est lui même en attente.

Les applications s'attendent donc mutuellement et sont toutes bloquées. C'est une situation **d'interblocage** (deadlock).

3. Ce constructeur automobile possède six sites de production qui échangent des documents entre eux. Les sites de production sont reliés entre eux par six routeurs A , B , C, D, E et F On donne ci dessous les tables de routage des routeurs A à F obtenus avec le protocole RIP :

Routeur A		Routeur B		Routeur C		Routeur D		Routeur E		Routeur F	
Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass
B	B	A	A	A	A	A	C	A	B	A	D
C	C	C	C	B	B	B	C	B	B	B	E
D	C	D	C	D	D	C	C	C	B	C	D
E	B	E	E	E	B	E	E	D	D	D	D
F	B	F	E	F	D	F	F	F	F	E	E

Déterminer à l'aide de ces tables le chemin emprunté par un paquet de données envoyé du routeur A vers le routeur F.

Corrigé

Le protocole RIP (*Routing Information Protocol*) est le premier algorithme de routage. Chaque routeur associe à chaque destination possible la plus courte distance en sauts (*hop*), c'est à dire en nombre de routeurs traversés pour aller au réseau.

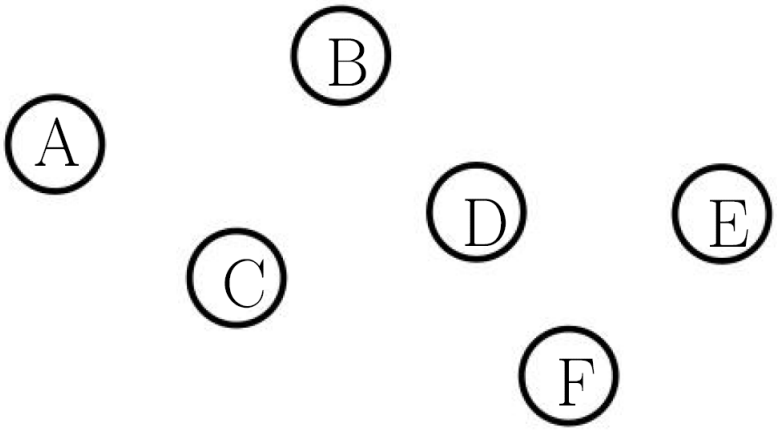
Le chemin emprunté par un paquet de données envoyé du routeur A vers le routeur F est d'après la table de routage :

$$\boxed{A \hookrightarrow B \hookrightarrow E \hookrightarrow F}$$

Routeur A		Routeur B		Routeur C		Routeur D		Routeur E		Routeur F	
Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass	Dest	Pass
B	B	A	A	A	A	A	C	A	B	A	D
C	C	C	C	B	B	B	C	B	B	B	E
D	C	D	C	D	D	C	C	C	B	C	D
E	B	E	E	E	B	E	E	D	D	D	D
F	B	F	E	F	D	F	F	F	F	E	E

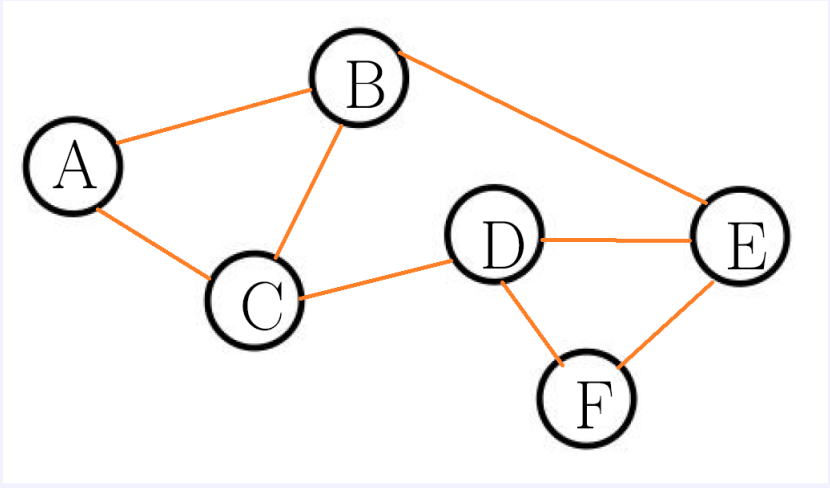
A -> B B -> E E-> F

4. On veut représenter schématiquement le réseau de routeurs à partir des tables de routage. Recopier sur la copie le schéma ci-dessous :



En s'appuyant sur les tables de routage, tracer les liaisons entre les routeurs.

Corrigé



Exercice 3. Les tableaux et sur la programmation de base en Python — 4 points

On rappelle que len est une fonction qui prend un tableau en paramètre et renvoie sa longueur. C'est-à-dire le nombre d'éléments présents dans le tableau.
Exemple : len([12, 54, 34, 57]) vaut 4.
Le but de cet exercice est de programmer différentes réductions pour un site de vente de vêtements en ligne.
On rappelle que si le prix d'un article avant réduction est de x euros,

- son prix vaut 0, 5x si on lui applique une réduction de 50%,
- son prix vaut 0, 6x si on lui applique une réduction de 40%,
- son prix vaut 0, 7x si on lui applique une réduction de 30%,
- son prix vaut 0, 8x si on lui applique une réduction de 20%,
- son prix vaut 0, 9x si on lui applique une réduction de 10%.

Dans le système informatique du site de vente, l'ensemble des articles qu'un client veut acheter, appelé panier, est modélisé par un tableau de ottants. Par exemple, si un client veut acheter un pantalon à 30,50 euros, un tee-shirt à 15 euros, une paire de chaussettes à 6 euros, une jupe à 20 euros, une paire de collants à 5 euros, une robe à 35 euros et un short à 10,50 euros, le système informatique aura le tableau suivant :

$$tab = [30.5, 15.0, 6.0, 20.0, 5.0, 35.0, 10.5]$$

1.

(a) Écrire une fonction Python **total_hors_reduction** ayant pour argument le tableau des prix des articles du panier d'un client et renvoyant le total des prix de ces articles.

Corrigé

On obtient :

```
# Dans l'éditeur Python
tab=[30.5, 15.0, 6.0, 20.0, 5.0, 35.0, 10.5]

def total_hors_reduction(tab):
    '''in  : tableau des prix
       out : prix total avant réduction'''
    somme=0
    for prix in tab:
        somme=somme+prix
    return somme
```

(b) Le site de vente propose la promotion suivante comme offre de bienvenue : 20% de réduction sur le premier article de la liste, 30% de réduction sur le deuxième article de la liste (s'il y a au moins deux articles) et aucune réduction sur le reste des articles (s'il y en a).

Recopier sur la copie et compléter la fonction Python **offre_bienvenue** prenant en paramètre le tableau **tab** des prix des articles du panier d'un client et renvoyant le total à payer lorsqu'on leur applique l'offre de bienvenue.

Corrigé

On obtient :

```
# Dans l'éditeur Python

def  offre_bienvenue ( tab ):
    """ tableau -> float """
    somme = 0
    longueur = len(tab)
    if longueur > 0 :
        somme = tab [0]*0.8 # réduction de 20% sur le 1er
    if longueur > 1 :
        somme = somme + tab [1]*0.7 # réd.  30% sur le 2e
    if longueur > 2 :
        for  i  in  range (2 , longueur ):
            somme =somme + tab[i]
    return somme
```

2. Lors de la période des soldes, le site de vente propose les réductions suivantes :
   - si le panier contient 5 articles ou plus, une réduction globale de 50%,
   - si le panier contient 4 articles, une réduction globale de 40%,
   - si le panier contient 3 articles, une réduction globale de 30%,
   - si le panier contient 2 articles, une réduction globale de 20%,
   - si le panier contient 1 article, une réduction globale de 10%.

   Proposer une fonction Python **prix_solde** ayant pour argument le tableau **tab** des prix des articles du panier d'un client et renvoyant le total des prix de ces articles lorsqu'on leur applique la réduction des soldes.

Corrigé

On obtient :

```
def prix_solde(tab):
    '''in  : tableau des prix
       out : prix total après réduction selon nb articles'''
    longueur = len ( tab )
    somme = total_hors_reduction(tab)
    if longueur>=5: # réduction globale de 50%
        somme = somme*0.5
    elif longueur == 4: # réduction globale de 40%
        somme = somme*0.6
    elif longueur == 3: # réduction globale de 30%
        somme = somme*0.7
    elif longueur == 2: # réduction globale de 20%
        somme = somme*0.8
    else:
        somme = somme*0.9  # réduction globale de 10%
    return somme
```

3.

(a) Écrire une fonction **minimum** qui prend en paramètre un tableau **tab** de nombres et renvoie la valeur minimum présente dans le tableau.

Corrigé

Il existe déjà la fonction **min** qui renvoie la valeur minimal d'un tableau de nombres. on propose deux versions possibles de la fonction proposée :

```
def minimum_v2(tab):
    '''in  : tableau des prix
       out : le prix min du tableau'''
    return min(tab)

def minimum(tab):
    '''in  : tableau des prix
       out : le prix min du tableau'''
    mini=tab[0]
    for prix in tab:
        if prix<mini:
            mini=prix
    return mini
```

(b) Pour ses bons clients, le site de vente propose une offre promotionnelle, à partir de 2 articles achetés, l'article le moins cher des articles commandés est offert. Écrire une fonction Python **offre_bon_client** ayant pour paramètre le tableau des prix des articles du panier d'un client et renvoyant le total à payer lorsqu'on leur applique l'offre bon client.

Corrigé

On obtient pour la fonction Python **offre_bon_client** :

```
def offre_bon_client(tab):
    '''in  : tableau des prix de longueur >=1
       out : total - article le moins cher'''
    longueur = len (tab)
    if longueur==1:
        return tab[0]
    else:
        return total_hors_reduction(tab)-minimum(tab)
```

4. Afin de diminuer le stock de ses articles dans ses entrepôts, l'entreprise imagine faire l'offre suivante à ses clients : en suivant l'ordre des articles dans le panier du client, elle considère les 3 premiers articles et offre le moins cher, puis les 3 suivants et offre le moins cher et ainsi de suite jusqu'à ce qu'il reste au plus 2 articles qui n'ont alors droit à aucune réduction.

   Exemple : Si le panier du client contient un pantalon à 30,50 euros, un tee-shirt à 15 euros, une paire de chaussettes à 6 euros, une jupe à 20 euros, une paire de collants à 5 euros, une robe à 35 euros et un short à 10,50 euros, ce panier est représenté par le tableau suivant :

$$tab = [30.5, 15.0, 6.0, 20.0, 5.0, 35.0, 10.5]$$

   Pour le premier groupe (le pantalon à 30,50 euros, le tee-shirt à 15 euros, la paire de chaussettes à 6 euros), l'article le moins cher, la paire de chaussettes à 6 euros, est offert. Pour le second groupe (la jupe à 20 euros, la paire de collants à 5 euros, la robe à 35 euros), la paire de collants à 5 euros est offerte. Donc le total après promotion de déstockage est 111 euros. On constate que le prix après promotion de déstockage dépend de l'ordre dans lequel se présentent les articles dans le panier.

(a) Proposer un panier contenant les mêmes articles que ceux de l'exemple mais ayant un prix après promotion de déstockage différent de 111 euros.

Corrigé

On peut proposer le tableau :

$$tab = \left[\boxed{6.0\ ,\ 20.0\ ,\ 5.0} , \boxed{35.0\ ,\ 10.5\ ,\ 30.5} , 15.0 \right]$$

On obtient alors :

— Pour les 3 premiers : $[6.0, 20.0, 5.0]$, on a $s_1 = 20 + 6 = 26$

— Pour les 3 suivants : $[35.0, 10.5, 30.5]$, on a $s_2 = 35 + 30.5 = 65.5$

— Au total :

$$\boxed{S = 26 + 65.5 + 15 = 106.5€}$$

(b) Proposer un panier contenant les mêmes articles mais ayant le prix après promotion de déstockage le plus bas possible.

Corrigé

Pour avoir le prix après déstockage le plus bas possible, il faut parvenir à offrir les articles les plus cher possible, et à payer celui le moins cher. On les classe par ordre décroissant :

$$35 > 30,5 > 20 > 15 > 10,5 > 6 > 5$$

$$tab = \left[\boxed{35\ ,\ 30.5\ ,\ 20} , \boxed{15\ ,\ 10.5\ ,\ 6} , 5 \right]$$

On obtient alors :

— Pour les 3 premiers : $[35\ ,\ 30.5\ ,\ 20]$, on a $s_1 = 35 + 30.5 = 65.5$

— Pour les 3 suivants : $[15\ ,\ 10.5\ ,\ 6]$, on a $s_2 = 15 + 10.5 = 25.5$

— Au total :

$$\boxed{S = 65.5 + 25.5 + 5 = 96€}$$

(c) Une fois ses articles choisis, quel algorithme le client peut-il utiliser pour modifier son panier afin de s'assurer qu'il obtiendra le prix après promotion de déstockage le plus bas possible ? On ne demande pas d'écrire cet algorithme.

Corrigé

— On peut utiliser un **algorithme de force brute**, c'est à dire un algorithme qui teste toutes les permutations possibles du tableau et choisi celle qui donne le prix de déstockage le plus bas possible.

— Ou on utilise un algorithme de tri décroissant, et on obtient le tableau qui donne le prix de déstockage minimal.

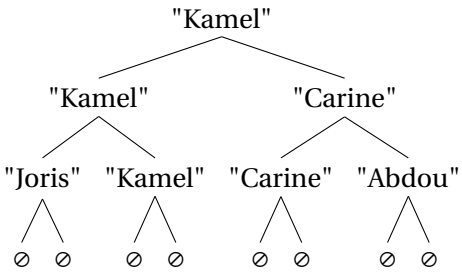
Compléments à l'exercice

On peut proposer une fonction Python **destockage** ayant pour paramètre le tableau des prix des articles du panier d'un client et renvoyant le total à payer lorsqu'on leur applique l'offre de déstockage. Une fonction récursive est assez naturelle ici :

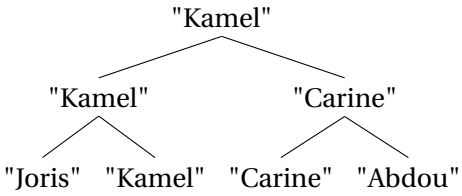
```
# Compléments
def destockage(tab):
    '''in  : tableau des prix
       out : prix total après destockage'''
    longueur=len(tab)
    if longueur<=2:
        return  total_hors_reduction(tab)
    else:
        return destockage(tab[3:longueur])+offre_bon_client(tab[0:3])
```

Exercice 4. les arbres binaires et leurs algorithmes associés — 4 points

La fédération de badminton souhaite gérer ses compétitions à laide dun logiciel. Pour ce faire, une structure arbre de compétition a été définie récursivement de la façon suivante : un arbre de compétition est soit l'arbre vide, noté ⊘, soit un triplet composé dune chaîne de caractères appelée valeur, d'un arbre de compétition appelé sous-arbre gauche et d'un arbre de compétition appelé sous-arbre droit. On représente graphiquement un arbre de compétition de la façon suivante :



Pour alléger la représentation dun arbre de compétition, on ne notera pas les arbres vides, l'arbre précédent sera donc représenté par larbre A suivant :



Cet arbre se lit de la façon suivante :

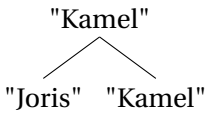
- 4 participants se sont affrontés : Joris, Kamel, Carine et Abdou. Leurs noms apparaissent en bas de l'arbre, ce sont les valeurs de feuilles de larbre.
- Au premier tour, Kamel a battu Joris et Carine a battu Abdou.
- En finale, Kamel a battu Carine, il est donc le vainqueur de la compétition.

Pour s'assurer que chaque finaliste ait joué le même nombre de matchs, un arbre de compétition a toutes ces feuilles à la même hauteur. Les quatre fonctions suivantes pourront être utilisées :

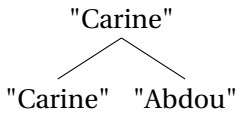
- La fonction **racine** qui prend en paramètre un arbre de compétition **arb** et renvoie la valeur de la racine.

  Exemple : en reprenant l'exemple darbre de compétition présenté ci-dessus, **racine**(A) vaut "Kamel".
- La fonction **gauche** qui prend en paramètre un arbre de compétition **arb** et renvoie son sous- arbre gauche.

  Exemple : en reprenant l'exemple darbre de compétition présenté ci-dessus, **gauche**(A) vaut l'arbre représenté graphiquement ci-après :



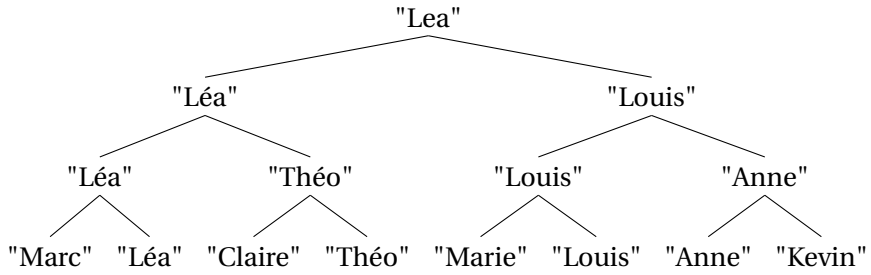
— La fonction **droit** qui prend en argument un arbre de compétition **arb** et renvoie son sous-arbre droit.
Exemple : en reprenant l'exemple darbre de compétition présenté ci-dessus, **droit**(A) vaut larbre représenté graphiquement ci-dessous :



— La fonction **est_vide** qui prend en argument un arbre de compétition et renvoie **True** si l'arbre est vide et **False** sinon.
Exemple : en reprenant l'exemple darbre de compétition présenté ci-dessus, **est_vide(A)** vaut **False**

Pour toutes les questions de l'exercice, on suppose que tous les joueurs dune même compétition ont un prénom différent.

1. (a) On considère l'arbre de compétition B suivant :



Indiquer la racine de cet arbre puis donner l'ensemble des valeurs des feuilles de cet arbre.

Corrigé

La racine de l'arbre est "Léa". L'ensemble des feuilles est donné par la liste : "Marc", "Lea", "Claire", "Theo", "Marie", "Louis", "Anne" et "Kevin".

(b) Proposer une fonction Python **vainqueur** prenant pour argument un arbre de compétition **arb** ayant au moins un joueur. Cette fonction doit renvoyer la chaîne de caractères constituée du nom du vainqueur du tournoi.
Exemple : **vainqueur**(B) vaut "Lea"

Corrigé

```
def vainqueur(arb):
    return racine(arb)
```

(c) Proposer une fonction Python **finale** prenant pour argument un arbre de compétition **arb** ayant au moins deux joueurs. Cette fonction doit renvoyer le tableau des deux chaînes de caractères qui sont les deux compétiteurs finalistes.
Exemple : **finale**(B) vaut ["Lea", "Louis"]

Corrigé

```
def finale(arb):
    return racine(gauche(arb)),racine(droit(arb))
```

2. (a) Proposer une fonction Python **occurrences** ayant pour paramètre un arbre de compétition **arb** et le nom dun joueur **nom** et qui renvoie le nombre doccurrences (d'apparitions) du joueur **nom** dans l'arbre de compétition **arb**.

   Exemple : **occurences**(B, "Anne") vaut 2.

Corrigé

Un parcours en profondeur est la méthode la plus simple.

```
def occurence(arb,nom):
    compt=0
    if est_vide(arb):
        return 0
    else:
        if racine(arb)==nom:
            compt=compt+1
        compt=compt+occurence(gauche(arb),nom)
        compt=compt+occurence(droit(arb),nom)
        return compt
```

   (b) Proposer une fonction Python **a_gagne** prenant pour paramètres un arbre de compétition **arb** et le nom dun joueur nom et qui renvoie le booléen **True** si le joueur nom a gagné au moins un match dans la compétition représenté par l'arbre de compétition **arb**.

   Exemple : **a_gagne**(B,"Louis") vaut **True**

Corrigé

La question précédente donne la solution.

```
def a_gagne(arb,nom):
    if occurence(arb,nom)>1:
        return True
    else:
        return False
```

3. On souhaite programmer une fonction Python **nombre_matchs** qui prend pour arguments un arbre de compétition **arb** et le nom d'un joueur **nom** et qui renvoie le nombre de matchs joués par le joueur **nom** dans la compétition représentée par l'arbre de compétition **arb**.

   Exemple : **nombre_matchs**(B,"Lea") doit valoir 3 et **nombre_matchs**(B,"Marc") doit valoir 1.

   (a) Expliquer pourquoi les instructions suivantes renvoient une valeur erronée. On pourra pour cela identifier le noeud de l'arbre qui provoque une erreur.

```
1   def   nombre_matchs ( arb , nom ):
2       """ arbre_competition , str-> int """
3       return  occurrences ( arb , nom )
```

Corrigé

Le nombre d'occurrence d'apparition d'un nom dans l'arbre n'est pas le nombre de matchs joués. Le noeud racine n'est pas un match joué. Il y a donc des joueurs qui sont comptés une fois de trop s'il y a un match gagnant.

(b) proposer une correction pour la fonction **nombre_matchs**.

Corrigé

Il suffit d'enlever 1 aux nombres d'occurrence quand il y a au moins un match de gagner.

```
def nombre_matchs(arb,nom):
    """ arbre_competition , str-> int """
    if a_gagne(arb,nom):
        return occurence(arb,nom)-1
    else:
        return occurence(arb,nom)
```

4. Recopier et compléter la fonction **liste_joueurs** qui prend pour argument un arbre de compétition **arb** et qui renvoie un tableau contenant les participants au tournoi, chaque nom ne devant figurer qu'une seule fois dans le tableau. L'opération + à la ligne 8 permet de concaténer deux tableaux.
Exemple : Si L1 = [4, 6, 2] et L2 = [3, 5, 1], l'instruction L1 + L2 va renvoyer le tableau [4, 6, 2, 3, 5, 1]

```
1   def  liste_joueurs ( arb ):
2       """ arbre_competition -> tableau """
3       if  est_vide ( arb ):
4           return .
5       elif ... and ... :
6           return  [ racine ( arb )]
7       else :
8           return  ...+ liste_joueurs ( droit( arb ))
```

Corrigé

```
def  liste_joueurs ( arb ):
      """ arbre_competition -> tableau """
      if  est_vide ( arb ):
          return []
      elif est_vide(gauche(arb)) and est_vide(droit(arb)):
          return  [ racine ( arb )]
      else :
          return liste_joueurs (gauche( arb ))+liste_joueurs(droit(arb))
```

Exercice 5. Notion de pile, de file et programmation de base en Python — 4 points

Les interfaces des structures de données abstraites Pile et File sont proposées ci-dessous. On utilisera uniquement les fonctions ci-dessous :

Structure de données abstraite : Pile
Utilise : Élément, Booléen
Opérations : • creer_pile_vide : ⊘ → Pile creer_pile_vide() renvoie une pile vide • est_vide : Pile → Booléen est_vide(*pile*) renvoie True si *pile* est vide, False sinon • empiler : Pile, Élément → ⊘ empiler(pile, element) ajoute *element* à la pile *pile* • depiler : Pile →, Élément depiler(pile) renvoie l'élément au sommet de la pile en le retirant de la pile

Structure de données abstraite : File
Utilise : Élément, Booléen
Opérations : • creer_file_vide : ⊘ → File creer_file_vide() renvoie une file vide • est_vide : File → Booléen est_vide(*file*) renvoie True si *file* est vide, False sinon • enfiler : file, Élément → ⊘ enfiler(file, element) ajoute *element* à la file *file* • defiler : file →, Élément defiler(file) renvoie l'élément au sommet de la file en le retirant de la file

1. (a) On considère la le F suivante :
enfilement ⟶ $\overline{"rouge"\ "vert"\ "jaune"\ "rouge"\ "jaune"}$ ⟶ défilement
Quel sera le contenu de la pile P et de la file F après lexécution du programme Python suivant?

```
1   P  =  creer_pile_vide ()
2   while   not( est_vide (F )):
3   empiler(P,  defiler(F))
```

Corrigé

La file F est vide et on obtient P=

"rouge"
"vert"
"jaune"
"rouge"
"jaune"

Créer une fonction **taille_file** qui prend en paramètre une file F et qui renvoie le nombre d'éléments qu'elle contient. Après appel de cette fonction la file F doit avoir retrouvé son état dorigine.

```
1   def taille_file ( F):
2   """ File - >  Int """
```

Corrigé

```
def taille_file(F):
    """File -> Int"""
    Q=creer_file_vide()
    long=0
    while not(est_vide(F)):
        long=long+1
        enfiler(Q,defiler(F))
    while not(est_vide(Q)):
        enfiler(F,defiler(Q))
    return long
```

2. Écrire une fonction **former_pile** qui prend en paramètre une file F et qui renvoie une pile P contenant les mêmes éléments que la file.

   Le premier élément sorti de file devra se trouver au sommet de la pile ; le deuxième élément sorti de la file devra se trouver juste en-dessous du sommet, etc.

   Exemple : si F ="rouge" "vert" "jaune" "rouge" "jaune" alors l'appel **former_pile(F)** va renvoyer la pile P ci-dessous :

P=

"jaune"
"rouge"
"jaune"
"vert"
"rouge"

Corrigé

```
def former_pile(F):
    Q=creer_pile_vide()
    T=creer_pile_vide()
    while not(est_vide(F)):
        empiler(Q,defiler(F))
    while not(est_vide(Q)):
        empiler(T,depiler(Q))
    return T
```

3. Écrire une fonction **nb_elements** qui prend en paramètres une file F et un élément *elt* et qui renvoie le nombre de fois où *elt* est présent dans la file F.

   Après appel de cette fonction la file F doit avoir retrouvé son état dorigine.

Corrigé

```
def nb_element(F,elt):
    compt=0
    Q=creer_file_vide()
    while not(est_vide(F)):
        a=defiler(F)
        if a==elt:
            compt=compt+1
        enfiler(Q,a)
    while not(est_vide(Q)):
        enfiler(F,defiler(Q))
    return compt
```

4. Écrire une fonction **verifier_contenu** qui prend en paramètres une file F et trois entiers : *nb_rouge*, *nb_vert* et *nb_jaune*.
   Cette fonction renvoie le booléen **True** si "rouge" apparaît au plus *nb_rouge* fois dans la file F, "vert" apparaît au plus *nb_vert* fois dans la file F et "jaune" apparaît au *plus nb_jaune* fois dans la file F. Elle renvoie **False** sinon. On pourra utiliser les fonctions précédentes.

Corrigé

```
def verifier_contenu(F,nb_rouge,nb_vert,nb_jaune):
    r=nb_element(F,"rouge")
    v=nb_element(F,"vert")
    j=nb_element(F,"jaune")
    if r>nb_rouge or  v>nb_vert or j>nb_jaune:
        return False
    else:
        return True
```

Fin du devoir