Bac Général Maths + NSI (Mathématiques et Numérique et sciences informatiques)

SUJET D'EXAMEN (SESSION OFFICIELLE)

Numérique et sciences informatiques - Session 2022

Sujet de l'examen national tel que publié pour la session indiquée. Source : sujets nationaux publiés par le ministère chargé de l'examen et les services académiques. Document mis à disposition par FormaV pour la révision.

Toutes les annales et leurs corrigés : https://bac-maths-nsi.fr/annales-sujets-et-corriges

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2022

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Mercredi 11 mai 2022

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 14 pages numérotées de 1/14 à 14 /14.

Le candidat traite au choix 3 exercices parmi les 5 exercices proposés

Chaque exercice est noté sur 4 points.

EXERCICE 1 (4 points)

Cet exercice composé de deux parties A et B, porte sur les structures de données.

Partie A : Expression correctement parenthésée

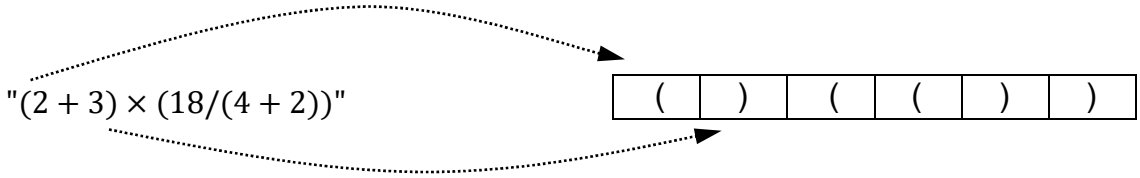
On veut déterminer si une expression arithmétique est correctement parenthésée.

Pour chaque parenthèse fermante ")" correspond une parenthèse précédemment ouverte "(".

Exemples :

- L'expression arithmétique "$(2+3)\times(18/(4+2))$" est correctement parenthésée.
- L'expression arithmétique "$(2+3)\times(18/(4+2$" est non correctement parenthésée.

Pour simplifier les expressions arithmétiques, on enregistre, dans une structure de données, uniquement les parenthèses dans leur ordre d'apparition. On appelle expression simplifiée cette structure.



1. Indiquer si la phrase « les éléments sont maintenant retirés (pour être lus) de cette structure de données dans le même ordre qu'ils y ont été ajoutés lors de l'enregistrement » décrit le comportement d'une file ou d'une pile. Justifier.

Pour vérifier le parenthésage, on peut utiliser une variable `controleur` qui :

- est un nombre entier égal à 0 en début d'analyse de l'expression simplifiée ;
- augmente de 1 si l'on rencontre une parenthèse ouvrante "(" ;
- diminue de 1 si l'on rencontre une parenthèse fermante ")".

Exemple : On considère l' expression simplifiée A : "( )(( ))"
Lors de l'analyse de l'expression A, `controleur` (initialement égal à 0) prend successivement pour valeur 1, 0, 1, 2, 1, 0. Le parenthésage est correct.

2. Écrire, pour chacune des 2 expressions simplifiées B et C suivantes, les valeurs successives prises par la variable `controleur` lors de leur analyse.
Expression simplifiée B : " (((( )( )"
Expression simplifiée C : "(( ))))("

3. L'expression simplifiée B précédente est mal parenthésée (parenthèses fermantes manquantes) car le `controleur` est différent de zéro en fin d'analyse. L'expression simplifiée C précédente est également mal parenthésée (parenthèse fermante sans parenthèse ouvrante) car le `controleur` prend une valeur négative pendant l'analyse.
Recopier et compléter uniquement les lignes 13 et 16 du code ci-dessous pour que la fonction `parenthesage_correct` réponde à sa description.

```
1 def parenthesage_correct(expression):
2    ''' fonction retournant True si l'expression arithmétique
3    simplifiée (str) est correctement parenthésée, False
4    sinon.
5    Condition: expression ne contient que des parenthèses
6    ouvrantes et fermantes '''

7   controleur = 0
8   for parenthese in expression: #pour chaque parenthèse
9      if parenthese == '(':
10        controleur = controleur + 1
11     else:# parenthese == ')'
12        controleur = controleur - 1
13        if controleur ...  : # test 1 (à recopier et compléter)
14           #parenthèse fermante sans parenthèse ouvrante
15           return False
16   if controleur ... : # test 2  (à recopier et compléter)
17      return True #le parenthésage est correct
18   else:
19      return False #parenthèse(s) fermante(s) manquante(s)
```

Partie B : Texte correctement balisé

On peut faire l'analogie entre le texte simplifié des fichiers HTML (uniquement constitué de balises ouvrantes <nom> et fermantes </nom>) et les expressions parenthésées :

Par exemple, l'expression HTML simplifiée :
" " est correctement balisée.

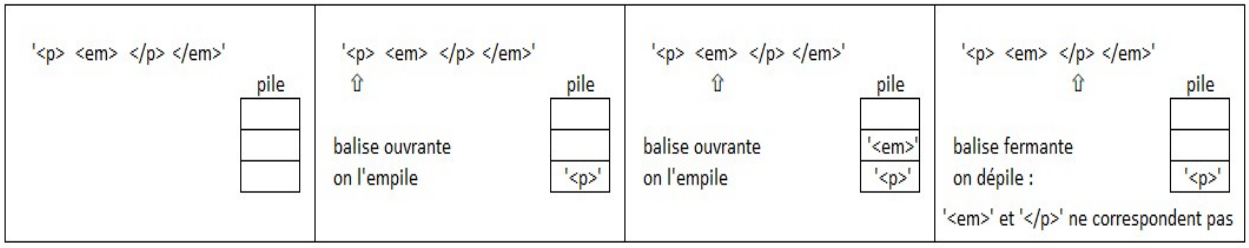
On ne tiendra pas compte dans cette partie des balises ne comportant pas de fermeture comme <br> ou <img …>.

Afin de vérifier qu'une expression HTML simplifiée est correctement balisée, on peut utiliser une pile (initialement vide) selon l'algorithme suivant :

On parcourt successivement chaque balise de l'expression :

- lorsque l'on rencontre une balise ouvrante, on l'empile ;
- lorsque l'on rencontre une balise fermante :
  - si la pile est vide, alors l'analyse s'arrête : le balisage est incorrect ,
  - sinon, on dépile et on vérifie que les deux balises (la balise fermante rencontrée et la balise ouvrante dépilée) correspondent (c'est-à-dire ont le même nom) si ce n'est pas le cas, l'analyse s'arrête (balisage incorrect).

Exemple : État de la pile lors du déroulement de cet algorithme pour l'expression simplifiée `"<p><em></p></em>"` qui n'est pas correctement balisée.



État de la pile lors du déroulement de l'algorithme

4. Cette question traite de l'état de la pile lors du déroulement de l'algorithme.

a. Représenter la pile à chaque étape du déroulement de cet algorithme pour l'expression `"<p><em></em></p>"` (balisage correct).

b. Indiquer quelle condition simple (sur le contenu de la pile) permet alors de dire que le balisage est correct lorsque toute l'expression HTML simplifiée a été entièrement parcourue, sans que l'analyse ne s'arrête.

5. Une expression HTML correctement balisée contient 12 balises.
Indiquer le nombre d'éléments que pourrait contenir au maximum la pile lors de son analyse.

EXERCICE 2 (4 points)

Cet exercice porte sur les bases de données.

On pourra utiliser les mots clés SQL suivants : **`SELECT`**, **`FROM`**, **`WHERE`**, **`JOIN`**, **`ON`**, **`INSERT`**, **`INTO`**, **`VALUES`**, **`UPDATE`**, **`SET`**, **`AND`**.

Nous allons étudier une base de données traitant du cinéma dont voici le schéma relationnel qui comporte 3 relations :

- la relation **`individu`** (id_ind, nom, prenom, naissance)
- la relation **`realisation`** (id_rea, titre, annee, type)
- la relation **`emploi`** (id_emp, description, #id_ind, #id_rea)

Les clés primaires sont soulignées et les clés étrangères sont précédées d'un #.
Ainsi **`emploi`**.id_ind est une clé étrangère faisant référence à **`individu`**.id_ind.
Voici un extrait des tables **`individu`** et **`realisation`** :

extrait de **`individu`**				extrait de **`realisation`**			
id_ind	nom	prenom	naissance	id_rea	titre	annee	type
105	'Hulka'	'Daniel'	'01-06-1968'	105	'Casino Imperial'	2006	'action'
403	'Travis'	'Daniel'	'10-03-1968'	325	'Ciel tombant'	2012	'action'
688	'Crog'	'Daniel'	'07-07-1968'	655	'Fantôme'	2015	'action'
695	'Pollock'	'Daniel'	'24-08-1968'	950	'Mourir pour attendre'	2021	'action'

1. On s'intéresse ici à la récupération de données dans une relation.

a. Écrire ce que renvoie la requête ci-dessous :

```
SELECT nom, prenom, naissance
FROM individu
WHERE nom = 'Crog';
```

b. Fournir une requête SQL permettant de récupérer le titre et la clé primaire de chaque film dont la date de sortie est strictement supérieure à 2020.

2. Cette question traite de la modification de relations.

a. Dire s'il faut utiliser la requête 1 ou la requête 2 proposées ci-dessous pour modifier la date de naissance de Daniel Crog. Justifier votre réponse en expliquant pourquoi la requête refusée ne pourra pas fonctionner.

Requête 1

```
UPDATE individu
SET naissance = '02-03-1968'
WHERE id_ind = 688 AND nom = 'Crog' AND prenom = 'Daniel';
```

Requête 2

```
INSERT INTO individu
VALUES (688, 'Crog', 'Daniel', '02-03-1968');
```

b. Expliquer si la relation **`individu`** peut accepter (ou pas) deux individus portant le même nom, le même prénom et la même date de naissance.

3. Cette question porte sur la notion de clés étrangères.

a. Recopier sur votre copie les demandes ci-dessous, dans leur intégralité, et les compléter correctement pour qu'elles ajoutent dans la relation **`emploi`** les rôles de Daniel Crog en tant que James Bond dans le film nommé 'Casino Impérial' puis dans le film 'Ciel tombant'.

```
INSERT INTO emploi
VALUES (5400, 'Acteur(James Bond)', ... );

INSERT INTO emploi
VALUES (5401, 'Acteur(James Bond)', ... );
```

b. On désire rajouter un nouvel emploi de Daniel Crog en tant que James Bond dans le film 'Docteur Yes'.
Expliquer si l'on doit d'abord créer l'enregistrement du film dans la relation **`realisation`** ou si l'on doit d'abord créer le rôle dans la relation emploi.

4. Cette question traite des jointures.

a. Recopier sur votre copie la requête SQL ci-dessous, dans son intégralité, et la compléter de façon à ce qu'elle renvoie le nom de l'acteur, le titre du film et l'année de sortie du film, à partir de tous les enregistrements de la relation emploi pour lesquels la description de l'emploi est `'Acteur(James Bond)'`.

```
SELECT ...
FROM emploi
JOIN individu ON ...
JOIN realisation ON ...
WHERE emploi.description = 'Acteur(James Bond)';
```

b. Fournir une requête SQL permettant de trouver toutes les descriptions des emplois de Denis Johnson (Denis est son prénom et Johnson est son nom). On veillera à n'afficher que la description des emplois et non les films associés à ces emplois.

EXERCICE 3 (4 points)

Cet exercice porte sur les représentations binaires et les protocoles de routage.

1. Une adresse IPv4 est représentée sous la forme de 4 nombres séparés par des points. Chacun de ces 4 nombres peut être représenté sur un octet.

a. Donner en écriture décimale l'adresse IPv4 correspondant à l'écriture binaire : 11000000.10101000.10000000.10000011

b. Tous les ordinateurs du réseau A ont une adresse IPv4 de la forme : 192.168.128._ _ _ , où seul le dernier octet (représenté par _ _ _) diffère. Donner le nombre d'adresses différentes possibles du réseau A.

2. On rappelle que le protocole RIP cherche à minimiser le nombre de routeurs traversés (qui correspond à la métrique). On donne les tables de routage d'un réseau informatique composé de 5 routeurs (appelés A, B, C, D et E), chacun associé directement à un réseau du même nom obtenues avec le protocole RIP :

Routeur A

Destination	Métrique
A	0
B	1
C	1
D	1
E	2

Routeur B

Destination	Métrique
A	1
B	0
C	2
D	1
E	2

Routeur C

Destination	Métrique
A	1
B	2
C	0
D	1
E	2

Routeur D

Destination	Métrique
A	1
B	1
C	1
D	0
E	1

Routeur E

Destination	Métrique
A	2
B	2
C	2
D	1
E	0

a. Donner la liste des routeurs avec lesquels le routeur A est directement relié.

b. Représenter graphiquement et de manière sommaire les 5 routeurs ainsi que les liaisons existantes entre ceux-ci.

3. Le protocole OSPF est un protocole de routage qui cherche à minimiser la somme des métriques des liaisons entre routeurs.

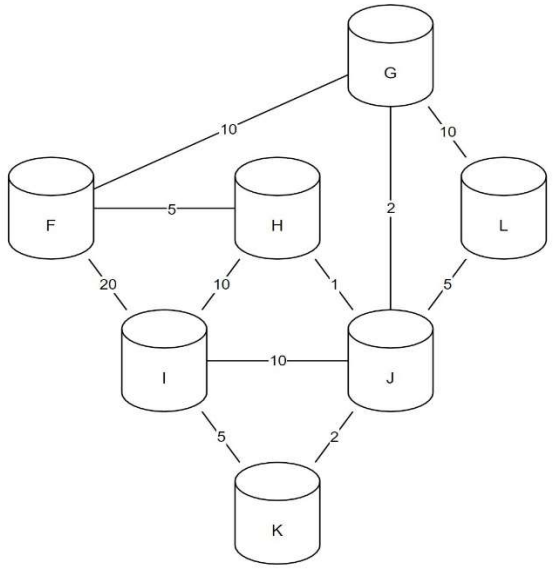
Dans le protocole de routage OSPF le débit des liaisons entre routeurs agit sur la métrique via la relation : $métrique = \frac{10^8}{débit}$ dans laquelle le débit est exprimé en bit par seconde (bps).

On rappelle qu'un kbps est égal à 10^3 bps et qu'un Mbps est égal à 10^6 bps.

Recopier sur votre copie et compléter le tableau suivant :

Débit	100 kbps	500 kbps	?	100 Mbps
Métrique associée	1 000	?	10	1

4. Voici la représentation d'un réseau et la table de routage incomplète du routeur F obtenue avec le protocole OSPF :



Routeur F

Destination	Métrique
F	0
G	8
H	5
I	
J	
K	
L	

Les nombres présents sur les liaisons représentent les coûts des routes avec le protocole OSPF.

a. Indiquer le chemin emprunté par un message d'un ordinateur du réseau F à destination d'un ordinateur du réseau I.
Justifier votre réponse.

b. Recopier et compléter la table de routage du routeur F.

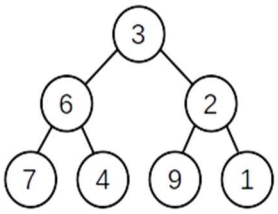
c. Citer une unique panne qui suffirait à ce que toutes les données des échanges de tout autre réseau à destination du réseau F transitent par le routeur G. Expliquer en détail votre réponse.

EXERCICE 4 (4 points)

Cet exercice, composé de deux parties A et B, porte sur le parcours des arbres binaires, le principe "diviser pour régner" et la récursivité.

Cet exercice traite du calcul de la somme d'un arbre binaire. Cette somme consiste à additionner toutes les valeurs numériques contenues dans les nœuds de l'arbre.

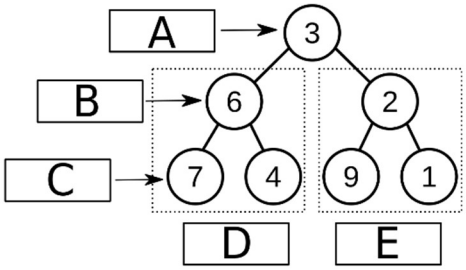
L'arbre utilisé *dans les parties A et B est le suivant :*



Partie A : Parcours d'un arbre

1. Donner la somme de l'arbre précédent. Justifier la réponse en explicitant le calcul qui a permis de l'obtenir.

2. Indiquer la lettre correspondante aux noms 'racine', 'feuille', 'nœud', 'SAG' (Sous Arbre Gauche) et 'SAD' (Sous Arbre Droit). Chaque lettre **A**, **B**, **C**, **D** et **E** devra être utilisée une seule fois.



Arbre avec les lettres à associer

3. Parmi les quatre propositions A, B, C et D ci-dessous, donnant un parcours en largeur d'abord de l'arbre, une seule est correcte. Indiquer laquelle.
Proposition A : 7 - 6 - 4 - 3 - 9 - 2 - 1
Proposition B : 3 - 6 - 7 - 4 - 2 - 9 - 1
Proposition C : 3 - 6 - 2 - 7 - 4 - 9 - 1
Proposition D : 7 - 4 - 6 - 9 - 1 - 2 – 3

4. Écrire en langage Python la fonction `somme` qui prend en paramètre une liste de nombres et qui renvoie la somme de ses éléments.
Exemple : `somme([1, 2, 3, 4])` est égale à 10.

5. La fonction `parcourir(arbre)` pourrait se traduire en langage naturel par :

```
parcourir(A):
    L = liste_vide
    F = file_vide
    enfiler A dans F
    Tant que F n'est pas vide
        défiler S de F
        ajouter la valeur de la racine de S dans L
        Pour chaque sous arbre SA non vide de S
            enfiler SA dans F
    renvoyer L
```

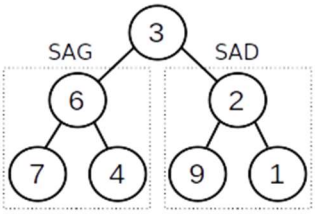
Donner le type de parcours obtenu grâce à la fonction `parcourir`.

Partie B : Méthode 'diviser pour régner'

6. Parmi les quatre propositions A,B, C et D ci-dessous, indiquer la seule proposition correcte.

En informatique, le principe diviser pour régner signifie :
Proposition A : diviser une fonction en deux fonctions plus petites
Proposition B : utiliser plusieurs modules
Proposition C : séparer les informations en fonction de leur types
Proposition D : diviser un problème en deux problèmes plus petits et indépendants.

7. L'arbre présenté dans le problème peut être décomposé en racine et sous arbres :



Indiquer dans l'esprit de 'diviser pour régner' l'égalité donnant la somme d'un arbre en fonction de la somme des sous arbres et de la valeur numérique de la racine.

8. Écrire en langage Python une fonction récursive `calcul_somme(arbre)`. Cette fonction calcule la somme de l'arbre passé en paramètre.

Les fonctions suivantes sont disponibles :

- `est_vide(arbre)` : renvoie `True` si `arbre` est vide et renvoie `False` sinon ;
- `valeur_racine(arbre)` : renvoie la valeur numérique de la racine de `arbre` ;
- `arbre_gauche(arbre)` : renvoie le sous arbre gauche de `arbre` ;
- `arbre_droit(arbre)` : renvoie le sous arbre droit de `arbre`.

EXERCICE 5 (4 points)

Cet exercice porte sur la Programmation Orientée Objet.

Les participants à un jeu de LaserGame sont répartis en équipes et s'affrontent dans ce jeu de tir, revêtus d'une veste à capteurs et munis d'une arme factice émettant des infrarouges.

Les ordinateurs embarqués dans ces vestes utilisent la programmation orientée objet pour modéliser les joueurs. La classe `Joueur` est définie comme suit :

```
class Joueur:
    def __init__(self, pseudo, identifiant, equipe):
        ''' constructeur '''
        self.pseudo = pseudo
        self.equipe = equipe
        self.id = identifiant
        self.nb_de_tirs_emis = 0
        self.liste_id_tirs_recus = []
        self.est_actif = True

    def tire(self):
        '''méthode déclenchée par l'appui sur la gachette'''
        if self.est_actif == True:
            self.nb_de_tirs_emis = self.nb_de_tirs_emis + 1

    def est_determine(self):
        '''methode qui renvoie True si le joueur réalise un
        grand nombre de tirs'''
        return self.nb_de_tirs_emis > 500

    def subit_un_tir(self, id_recu):
        '''méthode déclenchée par les capteurs de la
        veste'''
        if self.est_actif == True:
            self.est_actif = False
            self.liste_id_tirs_recus.append(id_recu)
```

1. Parmi les instructions suivantes, recopier celle qui permet de déclarer un objet `joueur1`, instance de la classe `Joueur`, correspondant à un joueur dont le pseudo est "Sniper", dont l'identifiant est 319 et qui est intégré à l'équipe "A":

Instruction 1 : `joueur1 = ["Sniper", 319, "A"]`

Instruction 2 : `joueur1 = new Joueur["Sniper", 319, "A"]`

Instruction 3 : `joueur1 = Joueur("Sniper", 319, "A")`

Instruction 4 :
```
joueur1 = Joueur{"pseudo":"Sniper",
                 "id":319, "equipe":"A"}
```

2. La méthode `subit_un_tir` réalise les actions suivantes :
Lorsqu'un joueur actif subit un tir capté par sa veste, l'identifiant du tireur est ajouté à l'attribut `liste_id_tirs_recus` et l'attribut `est_actif` prend la valeur `False` (le joueur est désactivé). Il doit alors revenir à son camp de base pour être de nouveau actif.

a. Écrire la méthode `redevenir_actif` qui rend à nouveau le joueur actif uniquement s'il était précédemment désactivé.

b. Écrire la méthode `nb_de_tirs_recus` qui renvoie le nombre de tirs reçus par un joueur en utilisant son attribut `liste_id_tirs_recus`.

3. Lorsque la partie est terminée, les participants rejoignent leur camp de base respectif où un ordinateur, qui utilise la classe `Base`, récupère les données.
La classe `Base` est définie par :
- ses attributs :
  - `equipe` : nom de l'équipe (`str`). Par exemple, "A" ,
  - `liste_des_id_de_l_equipe` qui correspond à la liste (`list`) des identifiants connus des joueurs de l'équipe,
  - `score` : score (`int`) de l'équipe, dont la valeur initiale est 1000 ;
- ses méthodes :
  - `est_un_id_allie` qui renvoie `True` si l'identifiant passé en paramètre est un identifiant d'un joueur de l'équipe, `False` sinon,
  - `incremente_score` qui fait varier l'attribut `score` du nombre passé en paramètre,
  - `collecte_information` qui récupère les statistiques d'un participant passé en paramètre (instance de la classe `Joueur`) pour calculer le score de l'équipe .

```
1 def collecte_information(self,participant):
2    if participant.equipe == self.equipe : # test 1
3       for id in participant.liste_id_tirs_recus:
4          if self.est_un_id_allie(id): # test 2
5             self.incremente_score(-20)
6          else:
7             self.incremente_score(-10)
```

a. Indiquer le numéro du test (**test 1** ou **test 2**) qui permet de vérifier qu'en fin de partie un participant égaré n'a pas rejoint par erreur la base adverse.

b. Décrire comment varie quantitativement le score de la base lorsqu'un joueur de cette équipe a été touché par le tir d'un coéquipier.

On souhaite accorder à la base un bonus de 40 points pour chaque joueur particulièrement déterminé (qui réalise un grand nombre de tirs).

4. Recopier et compléter, en utilisant les méthodes des classes `Joueur` et `Base`, les 2 lignes de codes suivantes qu'il faut ajouter à la fin de la méthode `collecte_information` :

```
........ #si le participant réalise un grand nombre de tirs
    ......... #le score de la Base augmente de 40
```