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

Asie - mai 2022 - sujet 2

Exercice 1 (Système d'exploitation - 4 points)
L'entreprise capNSI gère les contrats de ses clients en créant pour chacun d'eux un sous-dossier dans le dossier Contrats sur leur ordinateur central. Le système d'exploitation de cet ordinateur est une distribution linux. Quelques commandes de bases pour ce système d'exploitation sont rappelées en annexe 1 en fin de sujet.

Dans la console représentée sur la figure ci-dessous, on peut visualiser les répertoires (ou dossiers) à la racine de l'ordinateur central avec l'instruction `ls` :

```
gestion@capNSI-ordinateur_central:~$ ls
Bureau Documents Modèles Public
Téléchargements Contrats Images Musique
Vidéos
```

1. (a) Donner le nom de l'utilisateur et le nom de l'ordinateur correspondant à la capture d'écran précédente.
   (b) Ecrire les instructions permettant d'afficher la liste des dossiers clients du répertoire `Contrats` en partant de la situation ci-dessous :

```
gestion@capNSI-ordinateur_central:~$
```

Après une campagne de démarchage, l'entreprise a gagné un nouveau client : Monsieur Alan Turing. Elle souhaite lui créer un sous-dossier nommé `TURING_Alan` dans le dossier `Contrats`. De plus, elle souhaite attribuer tous les droits à l'utilisateur et au groupe et seulement la permission en lecture pour tous les autres utilisateurs. La commande `chmod` permet de le faire.

2. (a) Ecrire les instructions permettant de créer le sous-dossier `TURING_Alan` à partir du répertoire racine.
   (b) Ecrire l'instruction permettant d'attribuer les bons droits au sous-dossier `TURING_Alan`.

En Python, le module `os` permet d'interagir avec le système d'exploitation. Il permet de gérer l'arborescence des fichiers, des dossiers, de fournir des informations sur le système d'exploitation. Par exemple, le code ci-dessous, exécuté dans la console, permet de créer le sous-dossier `TURING_Alan` précédent :

```
>>> import os
>>> os.mkdir("Contrats/TURING_Alan")
>>> os.chmod("Contrats/TURING_Alan", 774)
```

L'entreprise dispose d'un tableau de nouveaux clients :

```
tab_clients = [('LOVELACE', 'Ada'),
               ('BOOLE', 'George'),
               ('VONNEUMANN', 'John'),
               ('SHANNON', 'Claude'),
               ('KNUTH', 'Donald')]
```

Elle souhaite automatiser le formatage des tableaux des nouveaux clients. Elle souhaite également automatiser la création et l'attribution des droits des dossiers portant les noms des nouveaux clients.

3. Ecrire une fonction `formatage` qui prend en paramètre un tableau `tab` de tuples `(Nom, Prenom)` des nouveaux clients et renvoie un tableau de chaînes de caractères. Par exemple, `formatage(tab_clients)` renvoie

```
['LOVELACE_Ada', 'BOOLE_George', 'VONNEUMANN_John', 'SHANNON_Claude', 'KNUTH_Donald']
```

4. Ecrire une fonction `creation_dossiers` qui prend en paramètre un tableau `tab` de chaînes de caractères et qui crée et modifie les droits des dossiers au nom de ces chaînes de caractères avec les mêmes droits que le sous-dossier `TURING_Alan`.

Exercice 2 (Arbres binaires de recherche - 4 points)

Un **arbre binaire de recherche** est un arbre binaire pour lequel chaque nœud possède une étiquette dont la valeur est supérieure ou égale à toutes les étiquettes des nœuds de son fils gauche et strictement inférieure à celles des nœuds de son fils droit. On rappelle que :

- sa taille est son nombre de nœuds ;
- sa hauteur est le nombre de niveaux qu'il contient.

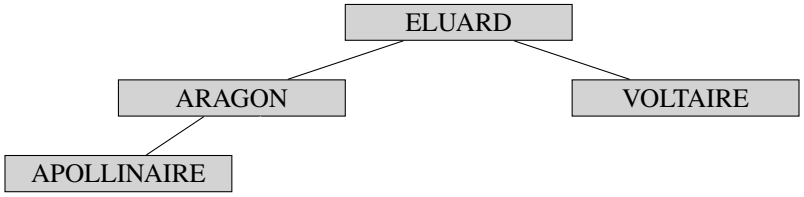
Un éditeur réédite des ouvrages. Il doit gérer un nombre important d'auteurs de la littérature. Pour stocker le nom des auteurs, il utilise un programme informatique qui les enregistre dans un arbre binaire de recherche.

L'arbre vide sera noté `Null` pour les algorithmes de cet exercice.
Si `A` est un nœud non vide, `valeur(A)` renvoie le nom de l'auteur ; `fils_gauche(A)` renvoie le fils gauche du nœud `A` et `fils_droit(A)` renvoie le fils droit du noeud `A`.

L'ordre alphabétique est utilisé pour classer le nom des auteurs. Par exemple, on a `APOLLINAIRE < BAUDELAIRE`.
Ainsi, pour tout nœud `A`, si `fils_gauche(A)` et `fils_droit(A)` ne sont pas `Null`, on a :

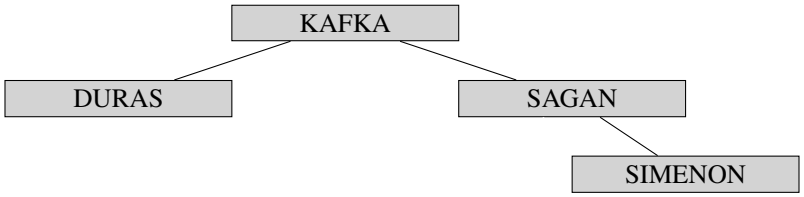
```
valeur(fils_gauche(A)) < valeur(A) < valeur(fils_droit(A))
```

Par exemple, l'arbre binaire `A1` suivant est un arbre binaire de recherche :



1. (a) Recopier et compléter l'arbre binaire de recherche précédent en insérant successivement dans cet ordre les noms suivants : DUMAS ; HUGO ; ZWEIG ; ZOLA.
   (b) Quelle est la taille de l'arbre obtenu ? Quelle est la hauteur de cet arbre ?
   (c) Plus généralement, si l'arbre est de hauteur h, quel est le nombre maximal d'auteurs enregistrés dans cet arbre en fonction de h ?

On définit ici l'équilibre d'un arbre binaire : il s'agit d'un nombre entier positif ou négatif. Il vaut 0 si l'arbre est vide. Sinon il vaut la différence des hauteurs des sous-arbres gauche et droit de l'arbre. Par exemple, on considère l'arbre suivant que l'on nommera `A2` :



Son équilibre vaut -1 car la hauteur de son sous-arbre gauche vaut 1, la hauteur de son sous-arbre droit vaut 2 et $1 - 2 = -1$.
Un arbre est dit équilibré si son équilibre vaut -1, 0 ou 1.
L'arbre précédent est donc équilibré.

2. Recopier et compléter l'arbre de ce dernier exemple avec les noms FLAUBERT, BALZAC, PROUST, SAND, WOOLF, COLETTE, CHRISTIE et AUDIARD quitte à modifier l'ordre d'insertion de manière à ce que cet arbre reste équilibré.

L'éditeur souhaite utiliser une fonction récursive `recherche_auteur` qui prend en paramètres un arbre binaire de recherche `ARB` et un nom d'auteur `NOM`. La fonction renvoie `True` si `NOM` est une étiquette de l'arbre `ABR` et `False` dans le cas contraire.

On donne la fonction suivante :

```
fonction mystere(ABR, t)
  si ABR = NULL alors
    renvoyer FAUX
  sinon si valeur(ABR) = t alors
    renvoyer VRAI
  sinon
    renvoyer mystere(fils_gauche(ABR),t) ou mystere(fils_droit(ABR),t)
  fin si
fin fonction
```

3. Que renvoie l'appel `mystere(A2,'SIMENON')` ? Justifier la réponse.

L'éditeur souhaite utiliser une fonction récursive `hauteur` qui prend en paramètre un arbre binaire `ABR` et renvoie la hauteur de cet arbre.

4. Ecrire un algorithme de la fonction `hauteur` qui prend en entrée un arbre binaire de recherche `ARB` et renvoie sa hauteur. On pourra avoir recours aux fonctions `MIN` et `MAX` qui renvoient respectivement la plus petite et la plus grande valeur des valeurs passées en paramètre.

Exercice 3 (Programmation générale - 4 points)

Le « jeu de la vie » se déroule sur une grille à deux dimensions dont les cases, qu'on appelle des « cellules », par analogie avec les cellules vivantes, peuvent prendre deux états distincts : « vivante » (= 1) ou « morte » (= 0).

Une cellule possède au plus huit voisins, qui sont les cellules adjacentes horizontalement, verticalement et diagonalement.

À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :

- ⋆ Règle 1 : une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît) ; sinon, elle reste à l'état « morte ».
- ⋆ Règle 2 : une cellule vivante possédant deux ou trois voisines vivantes reste vivante, sinon elle meurt.

Voici quatre exemples d'évolution du jeu de la vie appliquée à la cellule centrale :

1	1	0	devient	1	1	0	1	0	0	devient	1	0	0
0	**0**	0	par la	0	**1**	0	0	**1**	1	par la	0	**1**	1
0	0	1	règle 1	0	0	1	1	0	0	règle 2	1	0	0

0	0	0	devient	0	0	0	1	1	0	devient	1	1	0
1	**1**	0	par la	1	**0**	0	0	**1**	1	par la	0	**0**	1
0	0	0	règle 2	0	0	0	1	1	0	règle 2	1	1	0

Pour initialiser le jeu, on crée en langage Python une grille **de dimension 8×8**, modélisée par une liste de listes.

1. Initialisation du tableau :
   (a) Parmi les deux scripts proposés, indiquer celui qui vous semble le plus adapté pour initialiser un tableau de 0. Justifier votre choix.

Choix 1

```
ligne = [0,0,0,0,0,0,0,0]
jeu = []
for i in range(8) :
    jeu.append(ligne)
```

Choix 2

```
jeu = []
for i in range(8) :
    ligne = [0,0,0,0,0,0,0,0]
    jeu.append(ligne)
```

   (b) Donner l'instruction permettant de modifier la grille `jeu` afin d'obtenir la grille suivante :

```
>>> jeu
[[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0]]
```

2. (a) Ecrire en langage Python une fonction `remplissage` qui prend en paramètres un entier `n` et une grille `jeu`, et qui ajoute aléatoirement exactement `n` cellules vivantes dans le tableau `jeu`.
   (b) Quelles sont les préconditions de cette fonction pour la variable `n` ?

On propose la fonction en langage Python `nombre_de_vivants` qui prend en paramètres deux entiers `i` et `j` ainsi qu'une grille `jeu` et qui renvoie le nombre de voisins vivants de la cellule `jeu[i][j]` :

```
def nombre_de_vivants(i, j, jeu) :
    nb = 0
    voisins = [(i-1,j-1), (i-1,j), (i-1,j+1), (i,j+1), (i+1,j+1), (i+1,j),
               (i+1,j-1), (i,j-1)]
    for e in voisins :
        if 0 <= ... < 8 and 0 <= ... < 8 :
            nb = nb + jeu[...][...]
    return nb
```

3. Recopier et compléter les pointillés pour que la fonction réponde à la demande.
4. En utilisant la fonction `nombre_de_vivants` précédente, écrire en langage Python une fonction `transfo_cellule` qui prend en paramètres deux entiers `i` et `j` ainsi qu'une grille `jeu` et renvoie le nouvel état de la cellule `jeu[i][j]` (`0` ou `1`).

Exercice 4 (Bases de données et SQL - 4 points)

On souhaite gérer un club de tennis en ligne avec la possibilité de réserver un terrain à un créneau horaire. Le site ne gère que des réservations pour des matchs en simple. Voici la structure de la base de données :

⋆ Relation contenant l'ensemble des joueurs du club avec leurs identifiants :

joueurs				
id_joueur	nom_joueur	prenom_joueur	login	mdp
1	Dupont	Alice	alice	1234
2	Durand	Belina	belina	5694
3	Caron	Camilia	camilia	9478
4	Dupont	Dorine	dorine	1347

⋆ Relation précisant les matchs joués :

matchs					
id_match	date	id_creneau	id_terrain	id_joueur1	id_joueur2
1	2020-08-01	2	1	1	4
2	2020-08-01	3	1	2	3
3	2020-08-02	6	2	1	3
4	2020-08-02	7	2	2	4
5	2020-08-08	3	3	1	2
6	2020-08-08	5	2	3	4

⋆ Relation précisant les différents terrains.

terrains		
id_terrain	nom_terrain	surface
1	stade	terre battue
2	gymnase	synthétique
3	hangar	terre battue

⋆ Relation précisant les créneaux réservables.

creneaux	
id_creneau	plage_horaire
1	8h-9h
2	9h-10h
3	10h-11h
4	11h-12h
5	12h-13h
6	13h-14h
7	14h-15h
8	15h-16h
9	16h-17h
10	17h-18h
11	18h-19h
12	19h-20h

1. Clés primaires et étrangères :
   (a) Donner la clé primaire de la relation `matchs`.
   (b) La relation `matchs` a-t-elle une ou des clés étrangères ? Si oui quelles sont-elles ?
2. Par lecture et analyse des relations de la base de donnée.
   (a) Déterminer le jour et la plage horaire du match entre Durand Belina et Caron Camilia.
   (b) Déterminer le nom des deux joueurs qui sont les seuls à avoir joué dans le hangar.
3. Requêtes en langage SQL (on pourra s'aider de l'annexe 2).
   (a) Ecrire une requête qui renvoie les prénoms des joueurs dont le nom est Dupont.
   (b) Ecrire une requête qui modifie le mot de passe de Dorine Dupont, son nouveau mot de passe étant 1976.
4. Ecrire une requête permettant d'ajouter le nouveau membre Zora MAGID, dont le login est zora et le mot de passe 2021.
5. Ecrire une requête qui renvoie les jours où Alice joue.

Exercice 5 (Programmation générale - 4 points)

Les questions proposées sont indépendantes les unes des autres.

1. On considère la fonction `somme` qui prend en paramètre un entier `n` strictement positif et renvoie le résultat du calcul

$$1 + \frac{1}{2} + ... + \frac{1}{n}.$$

```
1 def somme(n) :
2     total = 0
3     for i in range(n) :
4         total = total + 1/i
5     return total
```

   Lors de l'exécution de `somme(10)`, le message d'erreur `"ZeroDivisionError: division by zero"` apparait. Identifier le problème et corriger la fonction pour qu'elle effectue le calcul demandé.

2. On considère la fonction `maxi` qui prend comme paramètre une liste `L` de nombres et renvoie le plus grand nombre de cette liste.

```
1 def maxi(L) :
2     indice = 0
3     maximum = 0
4     while indice <= len(L) :
5         if L[indice] > maximum :
6             maximum = L[indice]
7         indice = indice + 1
8     return maximum
```

   (a) Lors de l'exécution de `maxi([2, 4, 9, 1])` une erreur est déclenchée. Identifier et corriger le problème.

   (b) Le bug précédent est maintenant corrigé. Que renvoie à présent l'exécution de `maxi([-2, -7, -3])` ? Modifier la fonction pour qu'elle renvoie le bon résultat.

3. On souhaite réaliser une fonction qui génère une liste de `n` joueurs identifiés par leur numéro. Par exemple, on souhaite que l'appel `genere(3)` renvoie la liste `['Joueur 1', 'Joueur 2', 'Joueur 3']`.

```
1 def genere(n) :
2     L = []
3     for i in range(1, n+1) :
4         L.append('Joueur '+i)
5     return L
```

   L'appel `genere(3)` déclenche l'erreur suivante : `TypeError: can only concatenate str (not "int") to str`. Expliquer ce message d'erreur et corriger la fonction afin de régler le problème.

4. On considère la fonction `suite` qui reçoit un entier positif `n` et renvoie un entier.

```
1 def suite(n) :
2     if n == 0 :
3         return 0
4     else :
5         return 3+2*suite(n-2)
```

   (a) Quelle valeur renvoie l'appel `suite(6)` ?

   (b) Que se passe-t-il si on exécute `suite(7)` ?

5. On considère le code Python ci-dessous :

```
1 x = 4
2 L = []
3 def modif(x, L) :
4     x = x + 1
5     L.append(2*x)
6     return x, L
7 print(modif(x, L))
8 print(x, L)
```

   (a) Qu'affiche le premier `print` ?

   (b) Qu'affiche le second `print` ?

Annexe 1 (exercice 1)

(à ne pas rendre avec la copie)

Extrait des commandes de base linux :

- `ls` : permet d'afficher le contenu d'un répertoire
- `cd` : se déplacer dans l'arborescence (ex : `cd repertoire1`)
- `cp` : créer une copie d'un fichier (ex : `cp fichier1.py fichier2.py`)
- `mv` : déplacer ou renommer un fichier ou un répertoire (ex : `mv fichier.txt doss`)
- `rm` : effacer un fichier ou un répertoire (ex : `rm mon_fichier.mp3`)
- `mkdir` : créer un répertoire (ex : `mkdir nouveau`)
- `cat` : visualiser le contenu d'un fichier
- `chmod` : modifier les permissions d'un fichier ou d'un dossier.
  Pour un fichier, le format général de l'instruction est :

  ```
  chmod droits_user droits_group droits_other nom_fichier
  ```

  où `droits_user`, `droits_group` et `droits_other` indiquent respectivement les droits de l'utilisateur, du groupe et des autres et peuvent être :

  `+` : ajouter
  `-` : supprimer
  `r` : read
  `w` : write
  `x` : execute

  Exemple : `chmod rwx +r -x script.sh`

Annexe 2 (exercice 4)

(à ne pas rendre avec la copie)

⋆ **Types de données :**

`CHAR(t)`	Texte fixe de `t` caractères
`VARCHAR(t)`	Texte de `t` caractères variables
`TEXT`	Texte de $65\,535$ caractères maximum
`INT`	Nombre entier de -2^{31} à $2^{31}-1$ (signé) ou de 0 à $2^{32}-1$ (non signé)
`FLOAT`	Réel à virgule flottante
`DATE`	Date format `AAAA-MM-JJ`
`DATETIME`	Date et heure format `AAAA-MM-JJHH:MI:SS`

⋆ **Quelques exemples de syntaxe SQL :**

- Insérer des enregistrements :
```
INSERT INTO Table (attribut1, attribut2) VALUES(valeur1 , valeur2)
```

- Modifier des enregistrements :
```
UPDATE Table SET attribut1=valeur1, attribut2=valeur2 WHERE Selecteur
```

- Supprimer des enregistrements :
```
DELETE FROM Table WHERE Selecteur
```

- Sélectionner des enregistrements :
```
SELECT attributs FROM Table WHERE Selecteur
```

- Effectuer une jointure :
```
SELECT attributs FROM TableA JOIN TableB ON TableA.cle1=TableB.cle2 WHERE Selecteur
```