BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2022

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Jour 1

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 13 pages numérotées de 1/13 à 13/13.

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

Chaque exercice est noté sur 4 points.

Exercice 1

Thèmes abordés : algorithmique – chaînes de caractères – complexité.

Cet exercice propose l'étude d'un algorithme de détection de palindrome. On rappelle les définitions suivantes :

Définition 1 : un mot est un **palindrome** s'il peut se lire aussi bien dans les deux sens, par exemple le mot « kayak »

Définition 2 : un **variant de boucle** est une suite de valeurs d'entiers positifs strictement décroissante

On considère la fonction `palindrome1` qui renvoie un booléen et dont le paramètre `mot` est une chaîne de caractères de longueur `n`.

```
1|   Fonction palindrome1(mot) :
2|     Variables : i,j : ENTIER ; p : BOOLEEN
3|     i ← 0
4|     j ← longueur(mot)-1
5|     p ← Vrai
6|     tant que i≤j
7|       Si mot[i] ≠ mot[j]
8|         p ← Faux
9|       FinSi
10|      i ← i+1
11|      j ← j-1
12|    Fin tant que
13|    Renvoie p
```

1. Exécuter ligne après ligne cette fonction avec comme argument la chaîne de caractères "rotor" en recopiant le tableau suivant. L'étape 1 correspond à la première exécution de la boucle. Attention : toutes les colonnes ne sont pas forcément à remplir intégralement.

	Initialisation	Etape 1	Etape 2	Etape 3	Etape 4	…
`i ≤ j`						
`mot[i] ≠ mot[j]`						
`i`	0					
`j`	4					
`p`	Vrai					

2.

a. Combien de comparaisons de caractères sont réalisées pour cet algorithme avec le mot "rotor" ?

b. Combien de comparaisons de caractères sont réalisées pour un mot de longueur `n` dans le pire des cas ?

3. Montrer que la quantité `(j-i)` est un variant de la boucle `tant que`. En déduire que cette boucle se termine.

4. Combien de fois la boucle est-elle exécutée avec le mot "`routeur`" ? Proposer une amélioration de l'algorithme de la fonction `palindrome1` que l'on appellera `palindrome2` permettant d'éviter les tours de boucle inutiles. Justifier votre proposition.

Exercice 2

Thème abordé : bases de données

Un restaurant décide de créer son site de réservation en ligne pour son unique service du midi. Voici le schéma relationnel de la base de données imaginée par le concepteur du site. Elle est composée de 4 relations (tables) :

```
plat(id_plat, nom_plat, type_plat, prix_plat)
table_salle(num_table, nb_couvert_table, type_table)
client(num_client, nom_client, prenom_client, date_naiss_client,
mel_client, tel_client)
reservation(num_reserv, nb_pers_reserv, date_reserv, num_table, num_client)
```

Les chaînes de caractères dans cette base n'excèdent pas 100 caractères.
Le `num_client` et le `num_reservation` sont incrémentés automatiquement.
Voici un exemple possible d'enregistrement de la relation `plat` :

id_plat	nom_plat	type_plat	prix_plat
14	"brownie"	"Dessert"	6.35

1. Etude du schéma relationnel

a. Proposer pour chaque attribut de la relation `plat` son domaine (type), on pourra s'aider de l'annexe 1 : mémento SQL

b. Pour chacune des 4 relations, écrire les noms des clés primaires pouvant être utilisées.

c. Indiquer la ou les clés étrangères de la relation `reservation`. Préciser l'utilité d'une clé étrangère.

En vous aidant de l'ANNEXE 1 : memento SQL

2. Recherche et modification d'informations dans la base de données

a. Ecrire une requête SQL permettant d'afficher le nom des plats, leur type et leur prix.

b. Ecrire une requête SQL permettant d'afficher les noms de tous les plats proposés par le restaurant qui sont des desserts.

c. Les coordonnées d'un client ont été enregistrées :

num_client	nom_client	prenom_client	date_naiss_client	mel_client	tel_client
42	Martin	Fiona	1998-03-17	fmartin@laposte.net	"0605040302"

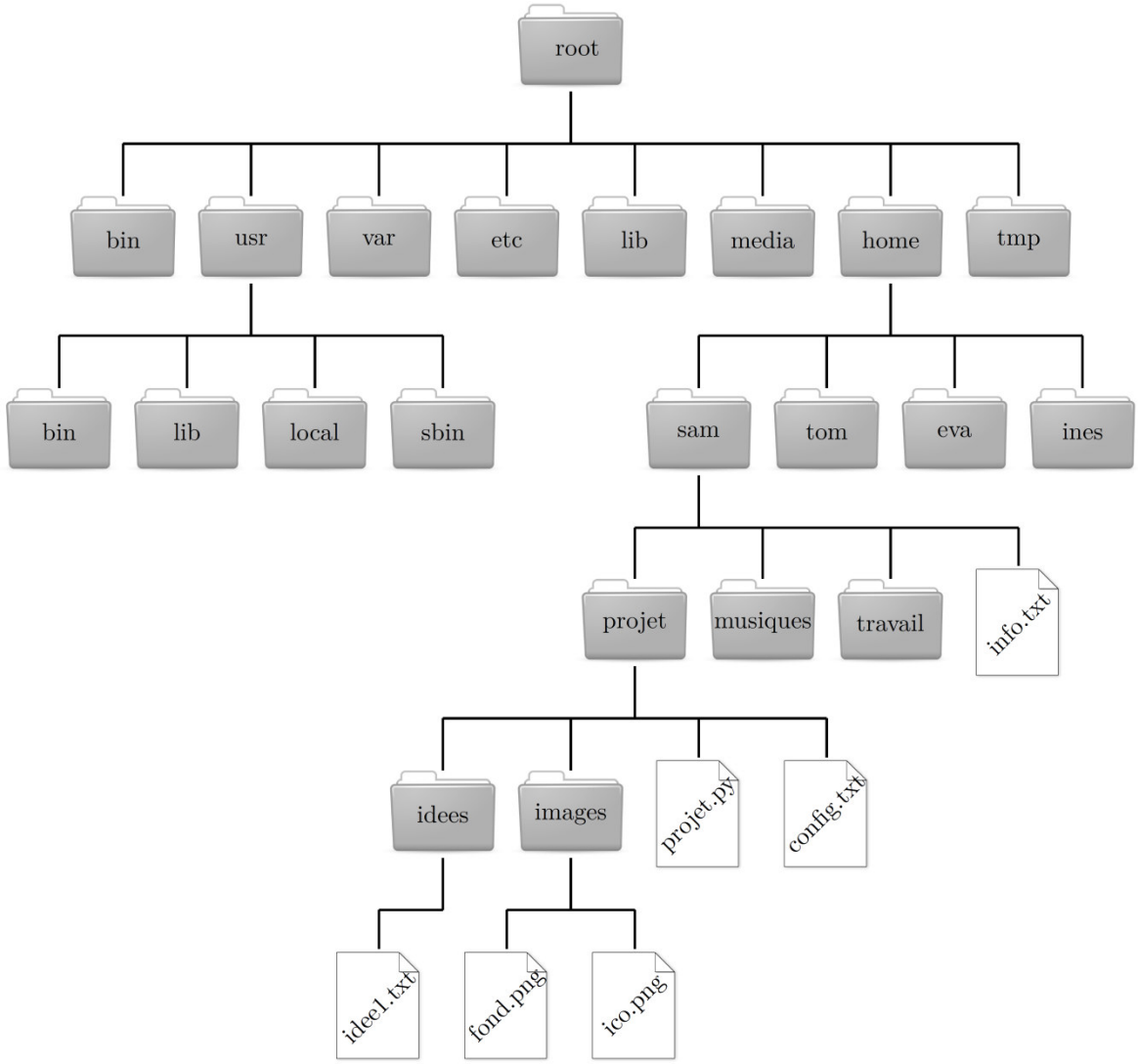
Ecrire la requête SQL permettant de modifier le numéro de téléphone de ce client en "0602030405".

d. Ecrire la requête SQL permettant d'afficher le nom de tous les clients ayant déjà mangé à la table n°13.

Exercice 3

Thème abordé : système d'exploitation

Nous avons l'arborescence ci-dessous sous un environnement Linux.



Samuel a pour nom d'utilisateur `sam`. Il a ouvert un terminal et le répertoire courant est le répertoire `musiques`. Pour tout l'exercice, on pourra tirer parti de l'annexe 2 répertoriant différentes commandes du système d'exploitation.

1. Ecrivez la ou les commande(s) qui permet(tent) de se déplacer du répertoire actuel `musiques` au répertoire `projet` :
   a. en utilisant un chemin relatif.
   b. en utilisant un chemin absolu.

2. Le répertoire courant est à présent le répertoire `sam`
   a. Ecrire la commande qui permet de lister le contenu du répertoire `projet`.
   b. Le fichier `config.txt` est protégé en écriture pour tous les utilisateurs. On souhaite modifier ce droit afin que l'utilisateur `sam` et lui seul puisse

modifier le contenu du fichier. Ecrire la commande permettant d'effectuer ce changement.

3. Le répertoire courant est toujours `sam`. L'utilisateur souhaite supprimer le répertoire `projet` en tapant l'instruction :

```
rm projet
```

Il constate que cette instruction ne fonctionne pas car ce répertoire n'est pas vide. *Finalement, il tape l'instruction :*

```
rm –R projet
```

où « R » signifie « récursif ». Le répertoire est finalement supprimé.

a. Pourquoi cette instruction fonctionne-t-elle, contrairement à la précédente ?

Les fichiers et dossiers ont été effacés dans cet ordre :

- fichier `idee1.txt`
- dossier `idees`
- fichier `fond.png`
- fichier `ico.png`
- dossier `images`
- fichier `projet.py`
- fichier `config.txt`
- dossier `projet`

b. Quel type de parcours a été réalisé par le système d'exploitation ?

4. On considère la fonction récursive suivante en langage Python :

```
def nb_fichiers(list_fich, i) :
  if i == len(list_fich) :
    return 0
  elif list_fich [i][0] == 'b' :
    return 1 + nb_fichiers(list_fich, i+1)
  else :
    return nb_fichiers(list_fich, i+1)
```

où `list_fich` est une liste contenant des noms de fichiers.

Indiquer ce que renvoie l'appel suivant en expliquant les étapes :

```
nb_fichiers(['nsi.bmp','banana.mp3','job.txt','BoyerMoore.py'],0)
```

Exercice 4

Thème abordé : programmation objet en langage Python

Un fabricant de brioches décide d'informatiser sa gestion des stocks. Il écrit pour cela un programme en langage Python. Une partie de son travail consiste à développer une classe `Stock` *dont la première version est la suivante :*

```
class Stock:
  def __init__(self):
    self.qt_farine = 0  # quantité de farine initialisée à 0 g
    self.nb_oeufs = 0   # nombre d'œufs (0 à l'initialisation)
    self.qt_beurre = 0  # quantité de beurre initialisée à 0 g
```

1. Écrire une méthode `ajouter_beurre(self, qt)` qui ajoute la quantité `qt` de beurre à un objet de la classe `Stock`.

On admet que l'on a écrit deux autres méthodes `ajouter_farine` et `ajouter_oeufs` qui ont des fonctionnements analogues.

2. Écrire une méthode `afficher(self)` qui affiche la quantité de farine, d'œufs et de beurre d'un objet de type `Stock`. L'exemple ci-dessous illustre l'exécution de cette méthode dans la console :

```
>>> mon_stock = Stock()
>>> mon_stock.afficher()
farine: 0
oeuf: 0
beurre: 0
>>> mon_stock.ajouter_beurre(560)
>>> mon_stock.afficher()
farine: 0
oeuf: 0
beurre: 560
```

3. Pour faire une brioche, il faut 350 g de farine, 175 g de beurre et 4 œufs. Écrire une méthode `stock_suffisant_brioche(self)` qui renvoie un booléen : `VRAI` s'il y a assez d'ingrédients dans le stock pour faire une brioche et `FAUX` sinon.
4. On considère la méthode supplémentaire `produire(self)` de la classe `Stock` donnée par le code suivant :

```
def produire(self):
```

```
    res = 0
    while self.stock_suffisant_brioche():
      self.qt_beurre = self.qt_beurre - 175
      self.qt_farine = self.qt_farine - 350
      self.nb_oeufs = self.nb_oeufs - 4
      res = res + 1
    return res
```

On considère un stock défini par les instructions suivantes :

```
>>> mon_stock=Stock()
>>> mon_stock.ajouter_beurre(1000)
>>> mon_stock.ajouter_farine(1000)
>>> mon_stock.ajouter_oeufs(10)
```

a. On exécute ensuite l'instruction

```
>>> mon_stock.produire()
```

Quelle valeur s'affiche dans la console ? Que représente cette valeur ?

b. On exécute ensuite l'instruction

```
>>> mon_stock.afficher()
```

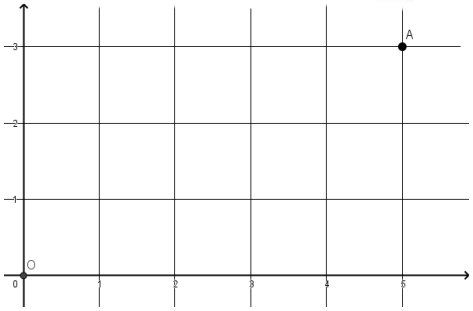
Que s'affiche-t-il dans la console ?

5. L'industriel possède `n` lieux de production distincts et donc `n` stocks distincts. On suppose que ces stocks sont dans une liste dont chaque élément est un objet de type `Stock`. Écrire une fonction Python `nb_brioches(liste_stocks)` possédant pour unique paramètre la liste des stocks et renvoie le nombre total de brioches produites.

Exercice 5

Thème abordé : programmation Python.

On considère un jeu de plateforme où un personnage se déplace dans un espace à deux dimensions. Pour cela, on autorise seulement deux déplacements élémentaires : de la gauche vers la droite ou du bas vers le haut. La longueur d'un déplacement correspond au nombre de déplacements élémentaires qui le constituent. Afin de représenter ces déplacements, on se place dans un repère où les coordonnées sont des nombres entiers positifs. Le personnage au début du jeu est situé à l'origine du repère de coordonnées (0,0) et il souhaite se rendre au point A de coordonnées (5,3).



Les lignes de code de cet exercice seront écrites en langage Python.

On décide de coder les déplacements élémentaires de la manière suivante :

- le caractère `'0'` représente un déplacement élémentaire vers la droite,
- le caractère `'1'` représente un déplacement élémentaire vers le haut.

Un déplacement de longueur `n` sera donc une chaîne de caractères composée de `n` caractères `'0'` ou `'1'`. Par exemple, un déplacement possible de O à A est `'00011001'` et sa longueur est 8.

1. On considère la fonction `mystere` ci-dessous.

```
1 | def mystere(dep):
2 |     x = 0
3 |     y = 0
4 |     for c in dep:
5 |       if c == '0':
6 |         x = x+1
7 |       else:
8 |         y = y+1
9 |     return [x, y]
```

a. Que renvoie `mystere('01110111')` ?

b. De manière plus générale, que renvoie la fonction `mystere` pour une chaîne représentant un déplacement donné ?

2. Ecrire une fonction `accessible(dep, arrivee)` de sorte qu'elle renvoie `True` si le déplacement `dep` se termine sur le point `arrivee` et `False` dans le cas contraire. Les types des paramètres sont donc :

- `dep` : une chaîne de caractères
- `arrivee` : une liste de deux entiers

Par exemple : `accessible('00110001', [5,3])` renvoie `True` alors que `accessible('01010111', [5,3])` renvoie `False`.

On rappelle que l'expression `str(randint(0,1))` utilisant la fonction `randint` de la bibliothèque `random` renvoie aléatoirement un caractère égal à `'0'` ou `'1'`.

On décide de trouver un déplacement de longueur 8 qui permette d'atteindre le point `arrivee`. Pour cela, on se propose de créer de manière aléatoire un déplacement de longueur 8, de tester si le point `arrivee` est accessible pour ce déplacement et de recommencer tant que ce n'est pas le cas.

3. Recopier et compléter la fonction `chemin` ci-dessous qui prend en paramètre les coordonnées du point `arrivee` et qui renvoie un déplacement de 8 pas qui permette d'atteindre le point `arrivee`. Quelles sont les préconditions sur le paramètre `arrivee` ?

```
1 | from random import randint
2 | def chemin(arrivee):
3 |   deplacement = '00000000'
4 |   while ..... :
5 |     .....
6 |     for k in range(8):
7 |       pas = str(randint(0,1))
8 |      ..... = deplacement + .....
9 |   return deplacement
```

4. La fonction `int(ch,2)` renvoie l'écriture décimale d'un nombre donné en binaire sous forme d'une chaîne de caractères `ch`. Par exemple `int('00000101',2)` renvoie `5`. Quelle est la plus grande valeur possible renvoyée par `int(chem,2)` si `chem` est un déplacement de longueur 8 permettant d'atteindre le point A de coordonnées (5,3) ?

Annexe 1 (exercice 2)

(à ne pas rendre avec la copie)

- **Types de données**

CHAR(t) VARCHAR(t) TEXT	Texte fixe de t caractères. Texte de t caractères variables. Texte de 65 535 caractères max.
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 DATETIME	Date format AAAA-MM-JJ 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
```

Annexe 2 (exercice 3)

(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 (`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`