ROYAUME DU MAROC
المملكة المغربية

,Ministère de l'Enseignement Supérieur,
de la Formation des Cadres et de la Recherche Scientifique

Présidence du Concours National Commun 2015
École Nationale Supérieure d'Électricité et de Mécanique

CONCOURS NATIONAL COMMUN

d'Admission dans les Établissements de Formation

d'Ingénieurs et Établissements Assimilés

Édition 2015

ÉPREUVE D'INFORMATIQUE

Filières : **MP/PSI/TSI**

Durée **2** heures

Cette épreuve comporte 07 pages au format A4, en plus de cette page de garde
L'usage de la calculatrice est **interdit**

Les candidats sont informés que la précision des raisonnements algorithmiques ainsi que le soin apporté à la rédaction et à la présentation des copies seront des éléments pris en compte dans la notation. Il convient en particulier de rappeler avec précision les références des questions abordées. Si, au cours de l'épreuve, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Remarques générales :

- *L'épreuve se compose de deux exercices et d'un problème.*
- *Toutes les instructions et les fonctions demandées seront écrites en langage Python.*
- *Les questions non traitées peuvent être admises pour aborder les questions ultérieures.*
- *Toute fonction peut être décomposée, si nécessaire, en plusieurs fonctions.*

Exercice 1. Thème : base de données

Nous allons nous intéresser à la gestion d'une compagnie de trains de marchandises en nous limitant à la gestions des trains. La base de données permettant la gestion des différents trains de cette compagnie sera constituée de 3 tables : *TRAIN, TRAJET, TYPE*.
La description de ces tables est la suivante :

TRAIN
immatriculation : entier
gare_attache : texte
id : entier

TRAJET
num_trajet : entier
immatriculation : entier
ville_dep : texte
ville_arr : texte
heure_dep : date

TYPE
id : entier
nom : texte
nb_place : entier

Une clé étrangère *id* est placée sur la table *TRAIN* et mise en relation avec la clé primaire *id* de la table *TYPE*.
Une clé étrangère *immatriculation* est placée sur la table *TRAJET* et mise en relation avec la clé primaire *immatriculation* de la table *TRAIN*.
Une clé primaire *num_trajet* est placée sur la table *TRAJET*.
Autre représentation :

TRAIN (**immatriculation**, gare_attache, *id*)

Exemple : *(2034, ''Fez'', 2)*

TRAJET (**num_trajet**, *immatriculation*, ville_dep, ville_arr, heure_dep, heure_arr)

Exemple : *(789,2034,''Rabat'',''Casablanca'',''09h00'',''10h20'')*

TYPE (**id**, nom, nb_place)

Exemple : *(2, ''TNR'', 440)*

Dans la représentation précédente les clefs principales sont en caractères gras et les clefs secondaires en italique et soulignés.

Vous pouvez utiliser le système de base de données que vous souhaitez. Cette base de données pourra être de type ***SQLite*** ou ***MySQL*** ou ***PostgreSQL*** etc.

Si vous utilisez une base de données ***SQLite***, le fichier « *gestion_trains.sqlite* » représentant cette base de données se trouvera dans le même répertoire que vos programmes ***Python*** (sous C:\CNC\) de cette compagnie ferroviaire.
Dans cet exercice la base de données est déjà créée : tables avec leurs contenus.

Question 1 :

Écrire une requête ***SQL*** donnant comme résultat l'heure de départ, l'heure d'arrivée et l'identifiant de tous les trains au départ de la ville de Fez.

Question 2 :

Modifier la requête précédente afin d'afficher, en plus des informations demandées, la gare d'attache de tous les trains toujours au départ de la ville de Fez. Attention on ne veut pas de doublon.

Question 3 :

Écrire une requête ***SQL*** donnant comme résultat le nombre de trajets au départ de Rabat par type de train.

Question 4 :

Écrire une requête ***SQL*** donnant le nom et la gare de départ de tous les trains dont l'heure de départ est comprise entre 7h et 13h.

Question 5 :

Écrire en ***Python*** le programme complet qui affichera le résultat de l'une des requêtes correspondant aux quatre premières questions. Vous pouvez utiliser le système de base de données que vous voulez, la création et le contenu des trois tables sont créés par défaut.

Exercice 2. Thème : calcul scientifique

On souhaite résoudre l'équation différentielle :

$$\frac{d^2y}{dt^2} = ay + b\frac{dy}{dt} \qquad (E)$$

avec : $a = -3.0$ et $b = -1.0$

Question 6 :

La fonction ***odeint()*** du module ***scipy.integrate*** a pour syntaxe :
scipy.integrate.odeint (fonction f, valeur initiale en a, subdivision de [a, b]).
La fonction ***odeint()*** ne permet de résoudre que des équations d'ordre 1.
Expliquer comment on peut transformer l'équation (E) en une équation du premier ordre.

Question 7 :

Le principe d'utilisation de la fonction ***odeint()*** permet d'obtenir une estimation numérique de la solution du problème de Cauchy : $\begin{cases} Y'(t)=F(Y(t),t) \\ Y(t_0)=Y_0 \end{cases}$

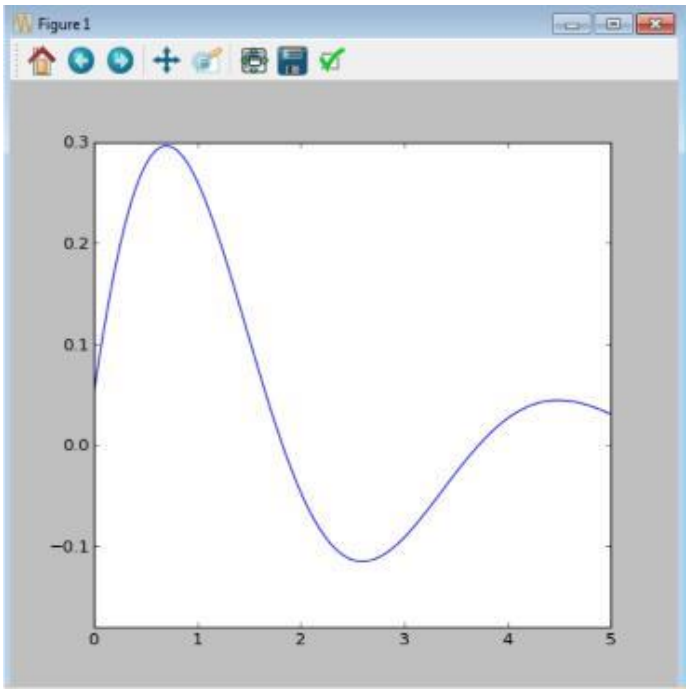
Écrire en ***Python*** une fonction ***F(z,t)*** qui prend en argument un vecteur colonne **z** composé de deux éléments, la variable ***t*** et qui retourne un tableau (*array* du module *numpy*) composé également de deux éléments.

La librairie ***numpy*** est très riche en fonctionnalités ***(np.arrange(), np.linspace(), np.array(), np.ones(), np.sin(), np.histogram(), np.concatenate(), …)***

Lors de l'implémentation de la fonction ***F()***, préciser bien à quoi correspondent les différents paramètres mis en œuvre.

Question 8 :

On souhaite avoir la représentation suivante de cette équation différentielle :



Pour la courbe représentée dans la ''Figure 1'' le nombre d'échantillons générés est égal à 100.

Implémenter dans le langage ***Python***, en utilisant la fonction ***odeint()*** citée précédemment, la formule ***(E)*** générant le graphique de la ''Figure 1''.

Attention vous devez pour les valeurs initiales qui vous manquent estimer leurs valeurs en observant la courbe de cette figure.

La librairie ***matplotlib*** permet de tracer toutes sortes de graphiques notamment avec la sous-librairie ***matplotlib.pyplot.***

Problème : Gestion des expressions postfixées

Remarque :
Les deux parties de ce problème sont indépendantes. La deuxième partie peut être faite en prenant les résultats de la première partie.
Par définition une expression est une suite de nombres associée à un ensemble d'opérateurs avec éventuellement un jeu de parenthèses ouvrantes et fermantes.
L'objet de ce problème est de gérer des expressions post fixées, expressions algébriques constituées d'éléments appelés opérandes et d'opérateurs. La position de l'opérateur par rapport aux opérandes indique le type d'expressions algébriques : infixées, préfixées, postfixées. La notation préfixée est souvent appelée notation polonaise car inventée par le mathématicien Jan Lukasiewicz et par opposition la notation postfixée est appelée notation polonaise inverse.
Il n'y aura aucune priorité particulière entre les différents opérateurs.
Exemple :
Notation infixée : *(1+5) *6*
Notation préfixée : *+ 1 6 **
Notation postfixée : *1 5 + 6 **
Autre exemple :
L'écriture postfixée de *(3+2) * 5* est *3 2+ 5 ** et celle de *3+(2 * 5)* est *3 2 5 * +*.

Question 9 :

Citer un intérêt d'utiliser la notation postfixée plutôt que la notation infixée.

Première partie

Afin de réaliser le traitement, c'est-à-dire le calcul d'une expression postfixée nous avons besoin d'un composant, une file d'attente, une ''pile'' de type FIFO (First In First Out).
Par définition une pile sera implémentée à l'aide d'une liste, le sommet de la pile sera toujours situé en début de liste. Vous devez dans cette première partie définir différentes fonctions capables d'utiliser une telle file d'attente qui n'aura pas de limite.
Afin que la pile puisse fonctionner correctement il faudra vérifier lors de la sortie d'un élément, que la file n'est pas vide. L'exception ***PileVideException*** est définie par défaut.
Pour l'implémentation de votre ***Pile*** vous devez utiliser les fonctions fournies dans l'annexe.

Question 10 :

Écrire une fonction ***initPile()*** afin d'initialiser une pile.

Question 11 :

Écrire la fonction ***estVide(pile)*** qui indique si une pile est vide.

Question 12 :

Écrire la fonction ***empiler(pile, elem)*** qui ajoutera à la file d'attente l'élément *elem*.

Question 13 :

Écrire la fonction ***depiler(pile)*** qui supprimera de la file d'attente le dernier élément entré et retournera cet élément.

Question 14 :

Écrire la fonction ***valeurSommet(pile)*** qui affichera l'élément se trouvant au sommet de la file d'attente. Cette fonction ne supprimera pas cet élément.

Question 15 :

Écrire la fonction ***hauteur(pile)*** qui affichera le nombre d'éléments se trouvant dans la pile.

Deuxième partie :

Principe de l'évaluation. Pour calculer une expression arithmétique codée en postfixée, il suffit :

- d'empiler les entiers successifs que l'on rencontre ;
- lorsque l'on détecte un opérateur, il faut
  - dépiler les 2 derniers entiers ;
  - leur appliquer l'opérateur ;
  - empiler le résultat ;
- enfin, lorsque l'on rencontre un point, le résultat du calcul doit être dépilé puis affiché.

Question 16 :

On souhaite tester si une expression est bien un nombre. La fonction ***estEntier()*** donnera comme résultat une valeur booléenne : True si l'expression est un nombre entier et False sinon. Attention, il n'est pas possible d'utiliser la fonction ***isinstance()***. La fonction ***estEntier()*** sera de style récursif.

```
>>>estEntier('12')
True
>>>estEntier(12)
True
>>>estEntier('1a4')
False
```

Question 17 :

Définir la fonction ***eval()*** qui sera constituée de trois paramètres, un opérateur (un caractère), suivi de deux opérandes (deux nombres entiers). Deux exceptions sont définies par défaut. L'exception ***OpNonValideException*** sera levée lorsqu'un opérateur ne sera pas valide et l'exception ***ArgNonValideException*** si un opérande est non valide. La liste des opérateurs utilisés est la suivante : +,-,*,/,. (le caractère point).

```
>>>eval('+', 1, 2)
3
>>>eval('+', '1', 2)
…
ArgNonValideException
```

Question 18 :

Définir la fonction ***evalue()*** qui aura comme arguments, l'expression postfixée à analyser. Cette fonction retournera la valeur du calcul correspondant à cette expression.

```
>>>evalue([7, 4, '+', '.'])
11
```

Question 19 :

On s'intéresse à présent aux expressions infixées, comme par exemple : 1+3*4 qui sera traduite par l'expression ['1','+','3','*','4','.']. Les parenthèses sont souvent utilisées afin de donner le sens que l'on souhaite aux différents calculs. Par exemple (1+3)*4 et 1+(3*4) donneront deux résultats différents. Dans cette question une expression pourra utiliser différents niveaux d'imbrication de parenthèses, comme par exemple (1+(5*(6+2))).

Définir la fonction ***estBienParenthesee()*** qui aura une expression (une liste de caractères) comme paramètre et retournera comme valeur une chaîne de caractère *Expression mal parenthésée* ou *Expression bien parenthésée*.

Question 20 :

Dans cette question les expressions infixées n'auront pas de parenthèses.

Définir la fonction ***transInfix()*** qui aura comme paramètre une expression infixée qui retournera une expression postfixée. On mettra en œuvre un mécanisme d'exception pour tester si l'expression fournie est vide ou mal formée.

```
>>>transInfix(['1', '+', '3', '*', '4', '+', '8', '.'])
['8', '4', '+', '3', '*', '1', '+', '.']
```

Question 21 :

En se limitant à un seul niveau de parenthèses, expliquez comment vous feriez pour transformer une expression infixée en une expression postfixée. On ne demande pas du code mais le mécanisme à mettre en œuvre.

```
>>>transInfix1SeulNiveau(['(', '1', '+', '2', ')', '*', '4', '.'])
['1', '2', '+', '4', '*', '.']
```

Question 22 :

Il y a différentes manières d'exprimer une expression postfixée. Les deux expressions suivantes sont totalement identiques, forme1 :['8', '4', '+', '3', '*', '1', '-', '.'] et
forme2 :['1', '3', '4', '8', '+', '*', '-', '.']

Définir la fonction ***transTotal()*** qui transforme une expression postfixée de type forme1 en une expression postfixée de type forme2.

```
>>>transTotal(expr)
['1', '3', '4', '8', '+', '*', '+', '.']
```

Question 23 :

Modifier la fonction précédente afin que le paramètre passé à ***transTotal()*** soit non pas une expression postfixée de type 1 mais une expression infixée.

```
>>>transTotal(['1', '+', '3', '*', '4', '+', '8', '.'])
['8', '4', '3', '1', '+', '*', '+', '.']
```

ANNEXE

str(p) : Retourne une chaîne contenant une représentation bien imprimable d'un objet ***p***. Pour les chaînes, cela renvoie la chaîne elle-même.

Opérations sur les listes :

Soit **L** un élément de type ***list***. La liste des méthodes des objets de type ***list*** est :
list.append(x) Ajoute un élément à la fin de la liste.
list.extend(L) Étend la liste en y ajoutant tous les éléments de la liste fournie.
list.insert(i, x) Insère un élément à la position indiquée. Le premier argument est la position de l'élément courant avant lequel l'insertion doit s'effectuer.
list.remove(x) Supprime de la liste le premier élément dont la valeur est x. Une exception est levée s'il n'existe aucun élément avec cette valeur.
list.index(x) Retourne la position du premier élément de la liste ayant la valeur x. Une exception est levée s'il n'existe aucun élément avec cette valeur.
list.count(x) Retourne le nombre d'éléments ayant la valeur x dans la liste.
list.sort(cmp=None, key=None, reverse=False) Trie les éléments de la liste.
list.reverse() Inverse l'ordre des éléments de la liste en place.

Figure 1 exercice 2