Langages de script
Examen

Numéro: _____

Consignes Vous avez 2 heures pour compléter l'examen. Si vous manquez de place pour inscrire vos réponses sur ce sujet (mais en principe vous ne devriez pas), indiquez-le et inscrivez la suite sur une copie. Merci de respecter les consignes suivantes :

- Tous les appareils électroniques doivent être éteints et rangés dans votre sac placé à l'avant de la salle.
- Seules 4 feuilles A4 recto verso de votre choix sont autorisées.
- Tous les programmes sont à écrire en PYTHON (ou en bash si précisé dans la question).
- Il n'est pas nécessaire de préciser les imports.

Exercice 1 : (5 points)

1. Écrire une fonction `dico` qui, étant donné un dictionnaire associant à des chaînes de caractères des listes d'entiers, renvoie un dictionnaire qui associe à chaque entier apparaissant dans les listes, le nombre de fois qu'il y apparaît. Par exemple,
`dico({"abb":[1,2,3],"k":[2,3,4],"jlk":[3,4,5]})` renvoie
`{1:1,2:2,3:3,4:2,5:1}`. Préférez un code court mais efficace.

2. Écrire une fonction `matrice` qui prend en paramètre une liste L et deux entiers, a et b, et retourne la matrice $a \times b$ composée de a lignes de b éléments de L. Par exemple,
`matrice([1,2,3,3,4,5], 2, 3)` retourne la matrice `[[1,2,3],[3,4,5]]` et
`matrice([1,2,3,3,4,5], 3, 2)` retourne `[[1,2],[3,3],[4,5]]`.
Si L ne contient pas $a \cdot b$ éléments, la fontion retourne `None`. Préférez une solution en une ligne.

3. On considère la définition de `b` suivante:

```
b = ["abcde","fghij","kl","m","nop"]
```

Indiquer ce qui est affiché par les commandes suivantes.

`print(2*b[0])`	
`print(b[0]+b[-1])`	
`print(b[0][:2:])`	
`print(b[1][::2])`	

Exercice 2 :
Le service de géolocalisation Nominatim de OpenStreetMap retourne des informations cartographiques sur les lieux. Voici un exemple de réponse à la requête `http://nominatim.openstreetmap.org/search.php?q=universite+Paris+Diderot&format=xml&extratags=1`

```
<searchresults timestamp="Wed, 10 May 17 10:44:54 +0000" attribution="Data © OpenStreetMap
contributors, ODbL 1.0. http://www.openstreetmap.org/copyright"
querystring="université Paris Diderot"
more_url="http://nominatim.openstreetmap.org/search.php?
q=universit%C3%A9+Paris+Diderot&extratags=1&format=xml">
<place place_id="108659787" osm_type="way" osm_id="188043991" place_rank="30"
lat="48.829058" lon="2.38093263926919" display_name="Tour Voltaire, Esplanade
Pierre Vidal Naquet, Gare, 13th Arrondissement, Paris,
Ile-de-France, 75013, France" class="amenity" type="university" importance="0.50855843404486" >
<extratags>
<tag key="wikipedia" value="fr:Université Paris VII - Diderot"/>
</extratags>
</place>
<place place_id="80106034" osm_type="way" osm_id="62377969" place_rank="30"
lat="48.82891495" lon="2.38340023397197" display_name="Condorcet B (Université Paris Diderot),
Quai Panhard et Levassor, Gare, 13th Arrondissement, Paris, Ile-de-France, 75013, France"
class="amenity" type="university" importance="0.311" >
<extratags/>
</place>
<place place_id="51913431" osm_type="node" osm_id="3775429263" place_rank="30"
lat="48.8294004" lon="2.3817559" display_name="Bétonsalon, Esplanade Pierre Vidal Naquet,
Gare, 13th Arrondissement, Paris, Ile-de-France, 75013, France" class="amenity"
type="arts_centre" importance="0.111" >
<extratags>
<tag key="website" value="http://www.betonsalon.net/"/>
</extratags>
</place>
</searchresults>
```

Supposons que la chaîne `rep` contienne le texte ci-dessus. (Le code doit aussi marcher sur d'autres exemples que celui-ci.) Pour chaque question, vous devez donner un appel à une méthode du module `re` qui permet de récupérer les informations demandées. Compléter le code ci-dessous pour que celui-ci affiche correctement les résultats.

1. La liste des codes postaux dans le texte. (Un code postal est composé de 5 chiffres)

2. La liste des valeurs associées à l'attribut `osm_type`. (Dans l'exemple, on devrait obtenir la liste `['way','way', 'node']` mais il peut y avoir d'autres valeurs possibles.)

3. Liste des coordonnées `lat, lon` dans la chaîne, ainsi que le nom du lieu associé (attribut `display_name`) et le type de lieu (attribut **`type`**).

```
# (1) votre code ici :
```

```
# (1) completer les parties soulignées:
print("Les codes postaux sont, dans l'ordre")

print(_____)
```

```
# (2) votre code ici :
```

```
# (2) completer les parties soulignées:
print("Les types sont, dans l'ordre")

print(_____)
```

```
# (3) votre code ici doit donner une valeur à la variable lieux:
```

```
# (3) completer les parties soulignées:
print("Les lieux sont les suivants")
for l in lieux :

    print ("Lieu: "+l.group(_____))

    print ("Coordonnees: "+l.group(_____)+","+l.group(_____))

    print ("Type: "+l.group(_____))
```

Exercice 3 :

Dans cet exercice, vous devez compléter le code de deux classes. La classe `Tache` permet de représenter les tâches à réaliser dans le cadre d'un projet. On donne au constructeur le nom de la tâche ainsi que la liste des noms des tâches qui doivent être complétées avant que celle-ci soit commencée. La classe `Projet` représente un ensemble de tâches à réaliser. La classe possède un itérateur qui retourne les tâches dans un ordre topologique suivant l'algorithme de Kahn. (Il n'est pas nécessaire de comprendre cet algorithme pour faire cet exercice mais on rappelle que les tâches sont dans l'ordre topologique lorsque les contraintes d'ordre entre les tâches sont respectées.)

On souhaite executer le code ci-après qui représente les étapes d'une recette de crêpes. Ce code est incomplet. L'affichage attendu est le suivant (noter que les tâches s'affichent dans l'ordre topologique) :

```
manger [apres cuire]
cuire [apres melanger], avant de, manger [après cuire]
mesurer -> casserlesoeufs -> melanger [apres mesurer, casserlesoeufs]
-> cuire [apres melanger] -> manger [apres cuire]
```

1. Souligner dans le code qui suit les endroits qui font appel à une méthode magique des classes `Tache` et `Projet`. Nommer à chaque fois la méthode magique appelée en précisant la classe.
2. Compléter le code de la classe `Tache` et la classe `Projet`.
   Votre code doit contenir toutes les méthodes nécessaires pour que le segment de code ci-dessus s'exécute correctement.

```
class Tache:
   def __init__(self, nom, prerequis):
       self.nom = nom
       self.prerequis = prerequis

class Projet:
   def __init__(self, L):
       # L est une liste d'objets de classe Tache
       self.taches = {x.nom : x for x in L}
       self._pred = dict()
   def _listpred(self):
       # Retourne un dictionnaire qui associe
       # à chaque nom de tâche la liste des noms
       # des tâches qui doivent la précéder
       return {x : [ y for y in self.taches
                     if self[y] < self[x] ]
                 for x in self.taches }
   def __iter__ (self):
       self._pred = self._listpred()
       return self
   def __next__ (self):
       # retourne les taches en ordre topologique
       next = None
       for x in self._pred:
           if len(self._pred[x])==0: next=x; break
       if next is None: raise(StopIteration)
       del self._pred[next]
       for tn in self._pred:
           if next in self._pred[tn]:
               self._pred[tn].remove(next)
       return self[next]

t1 = Tache("manger", ["cuire"])
t2 = Tache("cuire", ["melanger"])
t3 = Tache("melanger", ["mesurer","casserlesoeufs"])
t4 = Tache("casserlesoeufs",[])
t5 = Tache("mesurer",[])
crepes = Projet([t1,t2,t3,t4,t5])
print crepes["manger"]
if t2 < t1 : print (t2, "avant de", t1)
print(crepes)
```

Indiquer les méthodes magiques invoquées dans le code ci-contre

Fournir les méthodes manquantes ici en précisant la classe.

Exercice 4 :

1. On souhaite écrire un script bash `exec_python.sh` qui parcours les fichiers PYTHON (d'extension `.py`) d'un répertoire dont la référence est passée en paramètre et
   a. pour chaque fichier PYTHON , si la première ligne du fichier ne commence pas par *#!*, le script ajoute la première ligne *#!/usr/bin/env python3*. Par ailleurs le script ajoute le droit en exécution pour l'utilisateur sur le fichier.
   Par exemple:

```
$ls -l A
drwxr-xr-x  5 ubu  esque  4096 11 mai 11:29 E
-rw-r--r--  1 ubu  esque  1460 11 mai 11:30 bougrelas.txt
-rw-r--r--  1 ubu  esque   107 10 mai 16:52 mere.py
-rw-r--r--  1 ubu  esque    97 11 mai 11:25 pere.py

$head -n 2 A/mere.py
def mere():
  i=1

$head -n 2 A/pere.py
#!je suis le roi
def roi():


$./exec_python.sh A

$ls -l A
drwxr-xr-x  5 ubu  esque  4096 11 mai 11:29 E
-rw-r--r--  1 ubu  esque  1460 11 mai 11:30 bougrelas.txt
-rwxr--r--  1 ubu  esque   130 12 mai 10:02 mere.py
-rw-r--r--  1 ubu  esque    97 11 mai 11:25 pere.py


$head -n 2 A/mere.py
#!/usr/bin/env python3
def mere():

$head -n 2 A/pere.py
#!je suis le roi
def roi():
```

   Rappel: `head -n nb fic` *affiche les* `nb` *premières lignes du fichier de nom* `fic` *et* `head -c nb fic` *affiche les* `nb` *premiers caractères du fichier de nom* `fic`*.*

   b. Ensuite le script affiche la référence absolue du répertoire, ainsi que le nombre de fichiers modifiés. Sur l'exemple précédent, l'appel à la commande `exec_python.sh` produit en plus l'affichage suivant:

```
/user/info/ubu/A
1 fichier(s) modifié(s)
```

```
# code de exec_python.sh ici :
```

2. On souhaite en fait écrire un script PYTHON `rec_exec_python.py` permettant de réaliser l'action de `exec_python.sh` sur toute une arborescence dont la référence est passée en paramètre du script.
Écrire `rec_exec_python.py` qui doit appeler le script `exec_python.sh`.
Rappel: la fonction `os.system(com)` permet d'exécuter la commande bash représentée par la chaîne de caractères `com`.

```
# code de rec_exec_python.py ici :
```