Chaque question compte pour environ 1 point. Les questions 2a, 2b, et 2c comptent comme un tiers de question chacune. Aucun document (cours ou autre) n'est autorisé. Les calculatrices, ordinateurs... sont interdits.

```
c = dict([x.strip().split("=",1)
          for x in open("fichier").read().split("\n")
          if x and not x.startswith("#") and "=" in x])
```

1) Que fait ce bout de code ? (sans détailler ; il suffit de dire, par exemple : « il convertit un fichier en image JPEG », ou encore « il importe un fichier CSV dans une base SQL »)
2a) Expliquer à quoi sert l'appel à strip.
2b) Expliquer à quoi sert le second argument (1) de l'appel à split.
2c) Expliquer à quoi sert chacune des conditions qui suivent le "if" (il y en a 3 à expliquer).
3) Réécrire ce code sans utiliser la fonction dict, ni de « list comprehension » (la construction [f(x) for x in y if z]).
4) Réécrire ce code (avec éventuellement la fonction dict), mais sans « list comprehension » et sans boucle for/while. Il faudra donc utiliser lambda (entre autres). Si votre réponse à la question 3) correspond déjà à ces critères, inutile de donner une nouvelle réponse, marquez simplement "idem 3".

```
keys=map(lambda x:x[0],cursor.description)
return map(lambda r: dict(map(None,keys,r)),cursor.fetchall())
```

5) Réécrire ce code sans utiliser lambda.

6) On souhaite appeler la fonction « loop » indéfiniment toutes les 10 secondes. Le code suivant convient-il ? Si oui, le réécrire sans boucle while ; si non, expliquer pourquoi et donner une implémentation correcte en pseudo-code.

```
while 1:
    loop()
    time.sleep(10)
```

7) Écrire une fonction « transpose » transformant une liste de listes de la façon suivante : [[1,2,3],[4,5,6],[7,8,9],[10,11,12]] doit devenir [[1,4,7,10],[2,5,8,11],[3,6,9,12]]

8) Écrire une fonction « how_many_args » qui peut être appelée avec n'importe quel nombre d'arguments, et en indique le nombre ; par exemple :

```
>>> how_many_args(69,"toto",x="coin",y=None)
4
```

Un programme Python effectue les deux tâches suivantes :

- chargement d'une table SQL depuis une base de données (la table est composée de 4 champs VARCHAR(128), aucun n'est une clé, la table contient environ 100000 entrées) ;
- manipulation de ces données (rangement dans des tables de hachage, tri, etc.) et affichage.

Le programme étant très lent, on décide de le « profiler », et on découvre alors que ce n'est pas le traitement des données qui est long, mais le chargement depuis la base SQL. Plus précisément, une mystérieuse fonction nommée «typecast», contenue dans le module d'accès SQL, est appelée environ 400000 fois, et consomme 50% du temps CPU total. La requête SQL elle-même, exécutée dans une console SQL, prend un temps négligeable.

9) Donner une hypothèse sur ce que fait cette fonction (en la justifiant, si possible), et expliquer pourquoi, à votre avis, elle « rame ». C'est davantage le raisonnement que vous avancerez, que l'exactitude de la réponse, qui sera déterminant dans l'attribution des points.
10) Proposer une ou plusieurs solutions pour accélérer le programme (sachant que la deuxième phase, de manipulation des données, doit rester en Python, car la réécrire dans un autre langage serait trop coûteux).

```
def print_commandlist():
    for name in globals():
       if not name.startswith("action_"): continue
       docstring = globals()[name].__doc__
       if docstring==None: docstring = "<undocumented>"
       print name+": "+docstring
def action_hello():
    print "hello"
def action_goodbye():
    print "goodbye"
import sys
if len(sys.argv)<2: print_commandlist() ; sys.exit(0)
action_name = sys.argv[1]
action_func = globals()["action_"+action_name]
action_func(*sys.argv[2:])
```

11) Dire ce qui se passe si on lance le programme avec zéro argument.
12) Expliquer comment modifier action_hello et action_goodbye pour que la fonction print_commandlist affiche une information utile sur ces fonctions, au lieu de <undocumented>.
13) Modifier print_commandlist pour n'afficher que la première ligne de la docstring.
14) Expliquer ce qui se passe si on lance le programme avec un seul argument (détailler tous les cas possibles en fonction de l'argument).
15) Expliquer ce qui se passe si on lance le programme avec plusieurs arguments (détailler tous les cas possibles en fonction des arguments).

Qu'affichent ces programmes ? (un point par programme)

```
def melange(liste):
    liste=liste[::2]+liste[1::2]

liste = range(10) ; melange(liste) ; print liste
```

```
def pourcent(x,y):
    return x/y*100
print pourcent(1,4)
```

Les programmes qui suivent contiennent chacun une petite erreur. La corriger (un point par programme)

```
def augmentation_salaire(personne, oui_non):
  def p(x): print x ; return 1
  oui_non and p("%s est augmenté de 5%."%personne) or p("%s n'est pas augmenté."%personne)
augmentation_salaire("Bill", True)
```

```
cache = {}
def fibonacci(x):
    if x not in cache: cache[x] = fibonacci(x-1) + fibonacci(x-2)
    return cache[x]
print fibonacci(10)
```

```
import sys
print=sys.stderr.write
print "Ceci doit s'afficher sur stderr."
```

```
class nombre:
    def __init__(self, x): self.value = x
    def __add__(self, x): return nombre(self.value+x)
    def __str__(self): return str(self.value)
print nombre(4)+7
print 12+nombre(8)
```