Examen de Langages de script n° 2 : 2011/2012

- Durée de l'examen : 2h
- Vous devez éteindre et ranger vos téléphones.
- Les programmes sont à faire en PYTHON 3.
- L'annexe du sujet contient
  - des rappels de PYTHON ;
  - la documentation du module `re`.

Exercice 1 :

1. Écrire une fonction `alterner(l,s)` qui reçoit deux listes de même longueur en argument, et renvoie une liste contenant les éléments des deux arguments en alternance, c'est-à-dire dire `alterner([1,2,3],[4,5,6])` renvoie `[1,4,2,5,3,6]`, par exemple.
2. Écrire une fonction `flatten(l)` qui reçoit une liste de listes `l` en argument, et renvoie la concaténation des listes dans l. Par exemple, `flatten([[2,3,4],[3,5],[1]])` renvoie `[2,3,4,3,5,1]`.
3. Écrire une fonction `diffsym(a,b)` qui recoit deux ensembles `a,b` en argument, et renvoie leur *différence symétrique*, c'est-à-dire leur union sans leur intersection. Autrement dit, `x` est dans l'union symétrique de `a` et `b`, si x est dans l'un des deux, mais pas dans les deux en même temps. Par exemple, `diffsym({1,2,4,7},{1,3,4})` renvoie `{2,3,7}`.
4. Qu'imprime le programme suivant ?

```
a=sorted(list({x//2 for x in range(10)}))
b=sorted(list({x//2 for x in range(10,0,-1)}))
c=sorted(list({x//2 for x in range(10,0,-2)}))
print(a)
print(b)
print(c)
```

5. Qu'imprime le programme suivant ?

```
print(len({(a,b,c) for a in range(3) for b in range(4) for c in range(5)}))
```

6. Qu'imprime le programme suivant ?

```
def p(l):
    if len(l) == 0:
        return [[]]
    r = p(l[1:])
    x = l[0]
    s = []
    for m in r:
        for j in range(len(l)):
            s.append(m[:j]+[x]+m[j:])
    return s
print(p([0,1,2]))
```

   Expliquez votre réponse.

7. Écrire une fonction `csum(n)` qui étant donné un entier positif retourne la somme de ses chiffres. Par exemple pour `1234`, la fonction renvoie `10`.

Exercice 2 : Sécurité

Afin d'aider les utilisateurs à choisir des mots de passe plus sécurisés, on se propose de programmer un script testant la vulnérabilité du mot de passe.

Afin de contourner les attaques par force brute, un mot de passe doit vérifier certaines propriétés que nous allons tester.

Pour répondre à cet exercice, vous devez rédiger d'une part un module python nommé `security.py` et d'autre part donner les explications nécessaires à la compréhension de votre code.

Dans ce module vous pouvez ajouter tout le code que vous jugerez nécessaire (import de module, fonctions auxiliaires,...).

Ce module devra contenir les fonctions suivantes qui prennent en argument un mot de passe sous la forme d'une chaîne de caractère et renvoie un booléen `true` ou `false` si le mot de passe vérifie le test décrit :

mdp1 : Le mot de passe est assez long (plus de 8 caractères)

mdp2 : Le mot de passe contient au moins deux caractères non-alphanumériques.

mdp3 : Le mot de passe ne contient pas de mot issu d'un des lexiques suivants `capitales.txt`, `francais.txt`, `anglais.txt`, `annuaire.txt`. On supposera que ces lexiques sont des fichiers textes contenus dans le répertoire courant et dont chaque ligne contient un seul mot. On justifiera les structures de données choisies en terme d'efficacité.

mdp4 : Le mot de passe ne contient pas une variante d'un des mots issu des lexiques ci-dessus. La variante étant obtenue en remplaçant le caractère « a » par « @ » ; et la chaîne de caractères « et » par « & ». On pourra faire appel à la fonction précédente.

mdp5 : Le mot de passe ne contient pas le miroir d'un mot issu d'un des lexiques précédemment cité (le miroir de « passe » est « essap »).

Le module `security.py` se terminera par la partie principale du code qui ne sera exécutée que lorsque le module est lancé en ligne de commande à partir d'un terminal unix. Ce morceau de code demandera à l'utilisateur d'entrer son mot de passe et imprimera une note entre 1 et 5 selon le nombre de tests passés.

Exercice 3 : Mot de passe

Cet exercice propose d'écrire des scripts pour gérer les mots de passe.

Dans le système Unix, les identifiants des utilisateurs et leur mot de passe sont sauvegardés dans les fichiers `/etc/passwd` sous la forme suivante :

```
...
utilisateur1:x:1000:1000:Nom Utilisateur1,,,:/home/utilisateur1:/bin/bash
utilisateur2:x:1001:1001:Nom Utilisateur2,,,:/home/utilisateur2:/bin/bash
...
```

et `/etc/shadow` sous la forme suivante :

```
...
utilisateur1:$1$XopOFYH9$IfxyQwBe9b8tiyIkt2P4F/:13262:0:99999:7:::
utilisateur2:$1$vXGZLVbS$ElyErNf/agUDsm1DehJMS/:13261:0:99999:7:::
...
```

Les entrées du fichier `passwd` sont séparées par des « : », elles ont la signification suivante :

- identifiant de l'utilisateur
- entrée de spécification du mot de passe
- identifiant numérique de l'utilisateur
- identifiant numérique du groupe
- nom de l'utilisateur ou champ de commentaire
- répertoire personnel de l'utilisateur
- interpréteur de commandes, optionnel, de l'utilisateur.

Les entrées du fichier `shadow` ont la signification suivante :

- identifiant de l'utilisateur
- mot de passe chiffré (le « 1 » du début indique l'utilisation d'un chiffrement MD5. Le signe « * » indique que le compte ne peut pas se connecter)
- les autres données ne seront pas utilisées, elles correspondent à un nombre de jours, à partir du 1er janvier 1970.

Dans la suite, on supposera l'existence d'un module python `crypto.py` contenant une fonction de cryptage `chiffre` et une fonction de décryptage `dechiffre` que l'on pourra utiliser à condition de les avoir importés.

1. Écrire une fonction `lecture` qui prend en argument le nom d'un fichier et renvoie la liste de ses lignes. Écrire une fonction `ecriture` qui prend en entrée une liste de lignes et le nom d'un fichier et qui écrit les lignes à la suite des autres dans le fichier donné.
2. Afin de manipuler les fichiers `/etc/passwd` et `/etc/shadow/`, on propose d'utiliser un dictionnaire ayant comme clé l'identifiant de l'utilisateur et comme valeur un autre dictionnaire associant aux clefs suivantes :
   - `id_num` : identifiant numérique
   - `nom` : nom de l'utilisateur
   - `mdp` : mot de passe chiffré de l'utilisateur

   Écrire une fonction permettant d'initialiser ce dictionnaire. Écrire une fonction permettant de sauvegarder ce dictionnaire dans les fichiers `/etc/passwd` et `/etc/shadow/`. Certaines des données n'étant pas stockées dans le dictionnaire, vous pouvez les remplacer par une valeur arbitraire.
3. Écrire un script `adduser` qui permet de créer un nouvel utilisateur en demandant successivement
   - l'identifiant de l'utilisateur,
   - le nom de l'utilisateur,
   - le mot de passe de l'utilisateur, et une confirmation de ce mot de passe.

   Le script devra au fur et à mesure
   - vérifier que l'identifiant n'est pas déjà attribué,
   - vérifier que les deux mots de passe sont identiques,
   - vérifier que le mot de passe est un bon mot de passe,
   - déterminer un identifiant numérique qui n'est pas déjà attribué

   avant d'entrer les données du nouvel utilisateur dans le dictionnaire.
4. Écrire un script `passwd` qui permet à un utilisateur existant de modifier son mot de passe.
5. Décrire les opérations à effectuer afin que les commandes suivantes `./adduser` et `./passwd` lancées à partir d'un terminal lancent les scripts correspondants.

A Annexe

a) Rappel de quelques éléments de Python

– `range(i,j,l)` permet de parcourir les entiers de `i` à `j` exclu avec un pas de `l`.

```
>>> for i in range(3,-4,-1):
...     print(i)
...
3
2
1
0
-1
-2
-3
```

– L'opérateur `//` en python calcule la partie entière de la division. Par exemple, `5//2` donne `2`, et `3//4` a valeur `0`.

b) Module `re` - descriptif basé sur le livre de Harold Erbin

Syntaxe Les regex répondent à une syntaxe très codifiée et possèdent de nombreux symboles ayant un sens particulier. Pour débuter, tout caractère alphanumérique n'a pas de signification spéciale : A correspond simplement à la lettre A, 1 au chiffre 1, etc. Quant aux principaux symboles spéciaux, il sont :

`.` : désigne n'importe quel caractère ;

`^` : indique que le début de la chaine doit correspondre ;

`$` : indique que la fin de la chaine doit correspondre ;

`{n}` : indique que le caractère précédent doit être répété n fois.

`{n,m}` : indique que le caractère précédent doit être répété entre n et m fois.

`*` : le caractère précédent peut être répété aucune ou plusieurs fois. Par exemple, à ab* peuvent correspondre : a, ab, ou a suivi d'un nombre quelconque de b.

+ : le caractère précédent peut être répété une ou plusieurs fois. Par exemple, à ab+ correspond un a suivi d'un nombre quelconque de b.

? : le caractère précédent peut être répété zéro ou une fois. Par exemple, à ab ? correspondent ab et a.

L'antislash permet d'échapper tous ces caractères spéciaux. Les crochets [] permettent d'indiquer une plage de caractère, par exemple [e-h] correspondra à e, f, g ou h. Finalement, il reste quelques caractères spéciaux assez utiles :

`\w` : il correspond à tout caractère alphanumérique, c'est à dire qu'il est similaire à [a-zA-Z0-9_] ;

`\W` : il correspond à tout ce qui n'est pas un caractère alphanumérique ;

`\b` : il correspond à la frontière (début ou fin) d'un mot ;

`\d` : il correspond à tout caractère numérique, c'est à dire qu'il est similaire à [0-9] ;

`\D` : il correspond à tout ce qui n'est pas un caractère numérique.

Utilisation

`re.search(pattern, string)` Cherche le motif dans la chaîne passée en argument et retourne un `MatchObject` si des correspondances sont trouvées, sinon retourne `None`.

`re.split(pattern, string)` Découpe la chaine string selon les occurences du motif.

```
>>> re.split(r'\W', 'Truth is beautiful, without doubt.')
['Truth', 'is', 'beautiful', '', 'without', 'doubt', '']
```

`re.findall(pattern, string)` Retourne toutes les sous-chaines de `string` correspondant au motif.

`re.sub(pattern, repl, string)` Retourne la chaîne `string` où le motif a été remplacé par `repl`.