G. Hunault

Angers, décembre 2020

L3 Informatique

Décomposition, Conception et Réalisation d'Applications

1. Un peu d'expressions régulières

1.1 Indiquez le nombre de chaînes de caractères correspondant à l'expression régulière

```
^[Ll]a bal?le bleue?$
```

Vous détaillerez comment vous êtes arrivé(e) à ce nombre.

1.2 Listez par ordre alphabétique ces chaînes de caractères.

Pour mémoire, dans l'ordre alphabétique les majuscules sont situées AVANT les minuscules.

1.3 Donnez une expression régulière qui n'accepte que les deux chaînes Oui et Non.

1.4 Donnez une expression régulière qui n'accepte que les quatre chaînes O, Oui, N et Non.

2. Vive la ligne de commandes !

2.1 A l'aide des commandes find et grep, comment peut-on trouver le nombre de fichiers dont le nom contient la chaîne de caractères txt ?

2.2 A l'aide des commandes find et grep, comment peut-on trouver le nombre de fichiers dont le nom se termine par .txt ?

2.3 Expliquez ce que fait la commande

```
echo `find . | grep -c zip` fichiers
```

2.4 Quelle est la différence avec la commande

```
echo `find . | grep -c "\\.zip$" ` archives > listeArchives_`date +%Y-%m-%d`.txt
```

3. Un script de gestion d'archives

On voudrait écrire un script pour archiver automatiquement avec la commande zip soit seulement le répertoire courant, soit le répertoire courant et tous ses sous-répertoires. On voudrait également soit utiliser le nom du répertoire comme nom d'archive, soit ajouter un numéro incrémental au nom d'archive.

On suppose que le script se nomme sauveCd, que le premier paramètre est nommé cds et que le second paramètre est nommé remp. Voici les seuls quatre cas possibles d'utilisation correcte du script, exécutés dans un répertoire nommé devJava sachant que le prompt affiche `devJava @gh:>`. Il n'y a aucune question ici.

```
# 1. sauvegarde du répertoire et de ses sous-répertoires dans devJava.zip
#    commande exécutée au final : zip -u -r devJava.zip * -x *.zip

devJava @gh:> sauveCd tous remplacer

# 2. sauvegarde du répertoire seul dans devJava.zip
#    commande exécutée au final : zip -u devJava.zip * -x *.zip

devJava @gh:> sauveCd seul remplacer

# 3. sauvegarde du répertoire seul dans devJava003.zip
#    car devJava001.zip et devJava002.zip existent déjà
#    commande exécutée au final : zip -u devJava003.zip * -x *.zip

devJava @gh:> sauveCd seul incrementer

# 4. sauvegarde du répertoire et des ses sous-répertoires dans devJava004.zip
#    car devJava001.zip, devJava002.zip et devJava003.zip existent déjà
#    commande exécutée au final : zip -u -r devJava004.zip * -x *.zip

devJava @gh:> sauveCd tous incrementer
```

Pour ce qui suit, on décide que ce script Bash passera par l'appel d'une fonction Python.

Essayez de répondre aux questions suivantes. Vous pouvez répondre à la question 3.j même si vous n'avez pas répondu à la question 3.i (pour j > i).

3.1 Quelles sont les deux seules valeurs correctes possibles pour cds ?

3.2 Quelles sont les deux seules valeurs correctes possibles pour remp ?

3.3 Donnez le code d'une fonction Python (version 3) nommée SauveCd qui admet trois paramètres respectivement nommés nomRep, cds et remp et qui implémente seulement les fonctionnalités suivantes :

- la fonction importe le module nommé os ;
- la fonction renvoie systématiquement une chaîne de caractères correspondant à une ou plusieurs commandes **Linux** séparées par des points-virgule ;
- si nomRep (le nom du répertoire) est vide, ou si cds est vide ou si remp est vide, on le dit et on stoppe l'exécution ;
- si nomRep n'est pas un nom de répertoire valide, on le dit et on stoppe l'exécution ;
- si nomRep est la chaîne de caractères ".", on remplace nomRep par le nom du répertoire courant ;
- si cds est la chaîne de caractères "tous", on ajoute "-r" à la commande d'archivage ;
- si remp est la chaîne de caractères "remplacer", on met le nom du répertoire courant comme nom d'archive ;
- si remp est la chaîne de caractères "incrementer", on cherche le premier nom de fichier d'archive possible sachant que ce nom est composé du nom du répertoire et d'un nombre compris entre 1 et 999 formaté sur trois caractères avec des zéros de tête non significatifs (comme 001, 002...) ;
- si tous les paramètres sont OK, on renvoie la commande d'archivage.

3.4 Est-il possible de mettre des valeurs par défaut en Python (version 3) pour les paramètres dans la définition de la fonction ?

Si oui, quelles valeurs par défaut proposeriez-vous pour simplifier le code de la fonction ?

3.5 On suppose qu'on écrit un test unitaire pour vérifier ce que produit l'appel de la fonction SauveCd(".","oui","remplacer"). Que montrerait ce test pour notre fonction ?

3.6 On décide de remplacer la fonction SauveCd par une classe d'objets nommés reps (comme "répertoires") avec trois attributs nomRep, cds et remp et une méthode sauveCd avec deux paramètres nommés cds et remp.

Donnez le code de la définition de la classe reps. Vous implémenterez complètement les méthodes `__init__`, `getnomRep`, `setnomRep`, `setcds`, `getcds`, `setremp`. `setremp`. Pour la méthode sauveCd, vous fournirez juste un code qui renvoie la chaîne de caractères "archivage".

3.7 Quel est, selon vous, le meilleur choix : la fonction seule ou la classe d'objets ?

N'oubliez pas de détailler votre réponse.

3.8 Quel est, selon vous, le meilleur choix : écrire le script tout en Bash ou passer par une fonction Python ?

N'oubliez pas de détailler votre réponse.

3.9 Aurait-il été, selon vous, plus judicieux, d'écrire et d'appeler une fonction R plutôt qu'une fonction Python ? N'oubliez pas de détailler votre réponse.

4. Discussion culturelle

Essayez de répondre à la question suivante qui est sans doute *oxymorique* :

> Peut-on véritablement croire qu'il suffit de faire des copier/coller de codes trouvés sur Internet, même à partir de sites sérieux et de référence, pour prétendre savoir programmer efficacement ?

Votre réponse devra essayer de mettre en évidence votre culture naissante, votre recul et votre esprit de synthèse en matière de modélisation, de traitement de l'information et de la programmation. Vous pourrez, sans vous y limiter, prendre les cas concrets du développement Web en Python et/ou en PHP et/ou en Javascript ainsi que la programmation en Java, C++ ou R de structures de données complexes comme exemples.

Cette réponse devra faire 10 lignes au minimum, sans limite de maximum. On utilisera au moins 3 mots de 4 syllabes ou plus pour « *transmettre un contenu rédactionnel fort* ».

ESQUISSE DE SOLUTION

Question 1.1

Il y a deux possibilités pour `[Ll]`, deux possibilités pour `bal?le` et deux possibilités pour `bleue?`. Comme ces possibilités sont indépendantes, il y a en tout $2^3 = 8$ chaînes possibles.

Question 1.2

`[Ll]` correspond à L ou l, `bal?le` correspond à bale ou balle, `bleue?` correspond à bleu ou bleue. Sachant que l'espace est avant les majuscules, les huit chaînes sont :

```
La bale bleu
La bale bleue
La balle bleu
La balle bleue
la bale bleu
la bale bleue
la balle bleu
la balle bleue
```

Questions 1.3 et 1.4

L'expression régulière qui n'accepte que les deux chaînes Oui et Non est `^Oui|Non$`. Ou `^(Oui)|(Non)$` est acceptable aussi.

L'expression régulière qui n'accepte que les quatre chaînes O, Oui, N et Non est `^O(ui)?|N(on)?$` ou `^O|Oui|N|Non$`.

Question 2.1

La commande demandée est

```
find . | grep -c txt
```

Question 2.2

La commande demandée est

```
find . | grep -c "\\.txt$"
```

Question 2.3

La commande exécute ce qui est entre backticks puis affiche le mot fichiers. Cela donne le nombre de fichiers dont le nom contient zip suivi du mot fichiers. Voici un exemple d'exécution :

```
@ghchu> echo `find . | grep -c zip` fichiers
3246 fichiers
```

Question 2.4

La commande exécute ce qui est entre backticks puis affiche le mot archives avec une redirection de l'ensemble de l'affichage dans un fichier dont le nom commence par listeArchives_, qui contient ensuite la date au format indiqué puis qui se termine par .txt Voici un exemple d'exécution :

```
@ghchu> echo `find . | grep -c "\\.zip$" ` archives > listeArchives_`date +%Y-%m-%d`.txt
```

```
@ghchu> ls listeArchives_`date +%Y-%m-%d`.txt
listeArchives_2020-12-02.txt

@ghch> cat listeArchives_`date +%Y-%m-%d`.txt
33 archives
```

Question 3.1 et 3.2

Les deux seules valeurs correctes possibles pour cds sont tous et seul.

Les deux seules valeurs correctes possibles pour remp sont remplacer et incrementer.

Question 3.3

Voici le code demandé pour la fonction avec un auto-test :

```
 # -*- coding:latin1 -*-

def SauveCd(nomRep,cds,remp) : # version 1


  """

      sauvegarde du répertoire courant

        - incrémental ou fixe
        - avec ou sans sous-répertoires

      les 4 cas possibles sont

        SauveCd(nomRep,"seul","remplacer")
        SauveCd(nomRep,"tous","remplacer")
        SauveCd(nomRep,"seul","incrementer")
        SauveCd(nomRep,"tous","incrementer")

      la valeur "." est acceptable pour nomRep

  """

  import os

  if nomRep=="" :
    return " echo nomRep non renseigné "

  if cds=="" :
     return " echo cds non renseigné "

  if remp=="" :
     return " echo remp non renseigné "

  if nomRep=="." :
     nomRep =  os.path.basename(os.getcwd())

  # 1. on commence la commande zip

  cmd = "zip -u " # cela signfie update...
```

```
  # 2. on rajoute l'option de récursivité si on demande tous les sous-répertoires

  if cds=="tous" :
     cmd += " -r "

  # 3. on choisit le nom du fichier en fonction de l'option remp

  if remp=="remplacer" :
     cmd += nomRep + ".zip"
  else :
     # on adapte l'exemple vu dans le TP numéro 2
     numArchive = 1
     nomArchive = nomRep + ("%03d" % numArchive) + ".zip"

     while (os.path.isfile(nomArchive)) :
        numArchive += 1
        nomArchive = nomRep + ("%03d" % numArchive) + ".zip"
     # fin de tant que
     cmd += nomArchive
  # finsi

  # 4. on complète la commande zip

  cmd += " * -x *.zip "

  # si on arrive ici, on peut renvoyer la commande

  return cmd

# fin de fonction SauveCd

if __name__ == "__main__" :

   # il est possible de décommenter
   # les deux instructions suivantes pour plus de détails

   #    help(SauveCd)
   #    SauveCd()

   # comme la fonction renvoie une chaine de caractères,
   # on peut afficher ce qu'elle renvoie dans l'auto-test

   print( SauveCd("","","") )                # mauvais cas 1
   print( SauveCd(".","","") )               # mauvais cas 2
   print( SauveCd(".","seul","") )           # mauvais cas 3
```

```
   print( SauveCd(".","tous","remplacer") )   # bon cas 1
   print( SauveCd(".","seul","remplacer") )   # bon cas 2
   print( SauveCd(".","seul","incrementer") ) # bon cas 3
   print( SauveCd(".","tous","incrementer") ) # cas 4

# fin de l'auto-test
```

Le résultat de l'auto-test, légèrement aménagé, est le suivant

```
$gh> python3 decra2020-3.1.py

echo nomRep non renseigné
echo cds    non renseigné
echo remp   non renseigné

zip -u  -r Examens.zip    * -x *.zip
zip -u     Examens.zip    * -x *.zip
zip -u     Examens002.zip * -x *.zip
zip -u  -r Examens002.zip * -x *.zip
```

Question 3.4

Oui, il est possible de mettre des valeurs par défaut. Leur contenu serait sans doute celui des valeurs les plus courantes. Voici l'entête de la fonction précédente et celle de la nouvelle :

```
   # version 1 sans valeurs par défaut

   def SauveCd(nomRep,cds,remp) :

   # version 2 avec des valeurs par défaut

   def SauveCd(nomRep=".",cds="seul",remp="remplacer") :
```

Question 3.5

Le test unitaire montrerait qu'on n'a pas pris en compte d'autres valeurs que la chaine vide ou les seules valeurs correctes possibles pour les paramètres. Par exemple, oui n'est pas valide pour cds, ce qui n'est pas indiqué comme fonctionnalité.

Question 3.6

Voici le code demandé pour la classe d'objets avec un auto-test :

```
# -*- coding:latin1 -*-

class reps :

 """ une classe pour implémenter une sauvegarde avec zip """

 def __init__(self, nomRep=".",cds="seul",remp="remplacer") :
    self.nomRep = nomRep
    self.cds    = cds
    self.remp   = remp

 def sauveCd(self) :
     return "archivage"

 def getnomRep(self) :
     return self.nomRep

 def setnomRep(self,nomRep) :
     self.nomRep = nomRep

 def getcds(self) :
     return self.cds

 def setcds(self,cds) :
     self.cds = cds

 def getremp(self) :
     return self.remp

 def setremp(self,remp) :
     self.remp = remp

# fin de classe

if __name__ == "__main__" :

  monRep = reps(".","seul","remplacer")
  print( monRep.sauveCd() )

  print(" répertoire : ",monRep.getnomRep())
  monRep.setnomRep("oui")
```

```
  print(" répertoire : ",monRep.getnomRep())

  print(" répertoires : ",monRep.getcds())
  monRep.setcds("tous")
  print(" répertoires : ",monRep.getcds())

  print(" remplacement : ",monRep.getremp())
  monRep.setremp("incrementer")
  print(" remplacement : ",monRep.getremp())

# fin de l'auto-test
```

Le résultat de l'auto-test, légèrement aménagé, est le suivant

```
$gh> python3 decra2020-3.3.py

archivage
répertoire   :  .
répertoire   :  oui
répertoires  :  seul
répertoires  :  tous
remplacement :  remplacer
remplacement :  incrementer
```

Question 3.7

Pour une aussi petite application, les deux solutions proposées se valent sans doute. La fonction est cependant plus courte à écrire.

Question 3.8

Selon le niveau d'expertise, les deux solutions proposées se valent. Toutefois il est plus agréable de développer en Python si on ne maîtrise pas Bash et ses scripts.

Question 3.9

Rien ne justifie ici d'écrire une fonction R sauf à ne rien connaître ni à Python ni à Bash et à condition de savoir gérer les paramètres en ligne de commandes avec R.