**Système Unix et Langages de Scripts  
TP noté**

**1 Bash**

**1.1 Understanding**

1] Explain the code by means of executing and may be changing it. What does it do?

On définit une valeur max pour l’incrémentation, cette valeur est de 10000.

Ensuite on initialise notre nombre à la valeur voulue (ici 1), puis dans une boucle for on l’incrémente de 1 en 1.

Puis on effectue différents test. Tout d’abord on définit des variables locales via la commande (let « t1 = nr%5 », par exemple). Respectivement les variables t1, t2, t3 récupèrent les modulos 5, 7, 9 de la variable que l’on incrémente.

Une fois les variables initialisée, on effectue un test dans un if pour chaque variable, respectivement différent de 3, 4, 5.

Si un des tests est vrai, on continue le programme et on passe à la prochaine itération de la boucle for.

Si tous les tests sont faux, on arrive alors à l’instruction break, qui va nous faire sortir de la boucle for, et nous affichons alors la variable de cette boucle à la valeur correspondant au nombre d’incrémentation qu’elle a subi.

L’exit 0 permet de savoir si l’on a bien rentrer le programme et que tout s’est bien passé.

2] What does « break » do ?

Si aucune des conditions if du programme n’est vérifiée, on ne passe alors pas par l’instruction continue, et l’on arrive au break. Cette instruction va arrêter le déroulement de la boucle for et passer aux lignes de commandes suivant les accolades du for.

Pour résumer le break nous fais sortir de la boucle sans que celle-ci ai atteint sa condition de fin d’exécution.

3] What does « let » do ?

Le let initialise les variables t1, t2, t3 en leur affectant le résultat du calcul par modulo.

Donc let permet d’initialiser des variables. Il est aussi possible avec cette commande d’assigner des valeurs à plusieurs variables. On peut stocker dans ces variables une simple valeur ou le résultat d’un calcul arithmétique.

**1.2 Coding**

**2. Python**

**2.1 Understanding**

**2.2 Coding**

Dans un premier temps, on place le fichier « passwd » dans une variable nommée « fichier » permettant une lecture plus facile. De plus, on crée une liste « listLigne », servant à regrouper toutes les lignes de ce fichier, dans laquelle on place la première ligne afin d’initialiser la liste, et une variable « iterationLigne » permettant de savoir combien de ligne contient cette liste.

Nous rentrons ensuite dans une boucle « while » afin de placer toutes les lignes du fichier dans la liste précédemment initialisée. Dans ce « tant que », on test le contenu de la ligne placée dans la liste, si celle-ci est vide, on interrompt la boucle. Après ce test, on incrémente seulement la variable « iterationLigne », ceci permet d’exclure la dernière ligne de la liste ajoutée qui correspond à la chaîne vide.

Après cette extraction, on ferme tout d’abord le fichier. On demande ensuite à l’utilisateur de rentrer un nom que l’on récupéré dans une variable appelée « nom ». Pour la récupération des données associées à ce nom, on crée une variable « info », initialisée à la chaîne vide, destinée à contenir ces informations, et une variable « i » initialisée à 0 utilisée pour la prochaine boucle.

Ce nouveau « while » permet de parcourir la liste de lignes afin de savoir si le nom récupéré se trouve dans le fichier. Le test effectué vérifie si celui-ci correspond à la première valeur de la chaîne sur laquelle on se trouve, si c’est le cas, on affecte cette ligne splittée sur « : » à la variable « info » puis on sort de la boucle. Si le test est faux on incrémente i et on passe à la chaîne suivante dans la liste jusqu’à que toutes les lignes soient testées.

Lorsque l’on sort de la boucle, on test si info est vide. Si c’est le cas, cela signifie que le nom rentré par l’utilisateur ne se trouve pas dans le fichier, alors on précise l’information à ce dernier. Si la variable contient les informations de l’utilisateur (sous forme de tableau grâce au split), on affiche ces informations dans l’ordre demandé dans l’énoncé.

**3. Comparing Scripting Languages**

Pour la réalisation de ce programme nous utilisons le module de python « re » permettant de faire des opérations avec les expressions régulières. En effet c’est celui-ci qui nous permet d’effectuer les tests 2 et 3 décrit plus loin grâce à sa fonction « search » qui chercher si une expression régulière passée en paramètre et contenu dans la chaîne passée, elle aussi, en paramètre.

Dans ce programme, on rentre directement dans une boucle afin de permettre à l’utilisateur de tester plusieurs mots de passe tant que la variable « continuer » initialisée à 1 le reste, c’est-à-dire que l’utilisateur rentre 1 lorsque le mot de passe a fini d’être testé. C’est à l’intérieur de cette boucle que le réel intérêt de ce programme se trouve.

En effet, on retrouve une deuxième boucle qui continue tant que le mot de passe rentré n’est pas bon. Premièrement, dans cette boucle, nous demandons à l’utilisateur de rentrer un mot de passe que l’on place dans la variable « mdp ». Les variables suivantes vont servir pour les tests à venir, « aNbCar » contient le nombre de caractères du mot de passe, « aNombre » et « aCarSpec » contiennent des expressions régulières que doit respecter l’entrée de l’utilisateur. L’utilisation de « r. » devant ces expressions permet de ne pas mettre les « \ » devant les caractères spéciaux pour qu’ils soient compris, ceci permet de rendre les expressions régulières plus visibles.

Nous avons ensuite 3 tests. Le premier vérifie si la chaîne contient au moins 8 caractères. Le deuxième identifie si la chaîne contient au moins un chiffre. Et enfin, le troisième teste si le mot de passe contient au moins 1 caractère spécial. Ces tests sont passés les uns après les autres, si l’un deux n’est pas respecter on recommence la boucle et on indique a l’utilisateur en quoi le mot de passe n’est pas bon.