TP de Langages de script n° 8 : Scripts.

Exercice 1 :

Le but de cet exercice est de créer un script `convertisseur` qui convertit les températures et les longueurs entre le système anglo-saxon et le système français. On pourra l'utiliser dans un `shell` de la façon suivante :

```
convertisseur [-tl] [-s <systeme initial>] [nombres]

-t = temperature
-l = longueur
-s = introduit le systeme initial :
     A = anglo-saxon
     F = francais
```

1. Écrire les fonctions `celsius`, `fahrenheit`, `metre` et `yard` qui permettent de faire les différentes conversions.
2. Écrire une fonction `help` qui écrit dans le `shell` un message aidant à la bonne utilisation du script.
3. En vous aidant du module `sys` et de la méthode `argv`, écrire la fonction principale du script qui récupère les différents arguments, puis selon les cas, appelle une fonction de conversion sur l'argument adéquat ou appelle la fonction d'aide.

Exercice 2 :

Le but de cet exercice est de créer un script `capitales` qui permet de tester ses connaissances sur les capitales du monde. On pourra l'utiliser dans un `shell` de la façon suivante :

```
capitales -v ville
    -> le script demande a l'utilisateur le nom du pays dont la capitale
    est ville, puis verifie la reponse
capitales -p pays
    -> le script demande a l'utilisateur le nom de la capitale du pays,
    puis verifie la reponse
capitales -n
    -> le script demande au hasard des villes ou des pays, le joueur peut
    decider de continuer ou d'arreter, a la fin il obtient une
    statistique sur le nombre de bonne reponse.
```

1. Écrire une fonction qui convertit un fichier contenant des pays et leur capitale en un ou des objets (liste, dictionnaire, ensemble, ... au choix) adaptés à la situation. Tester cette fonction sur le fichier `capitales.txt` que vous trouverez sur Didel.
2. Écrire une fonction pour chacune des options `-v, -p, -n` (on s'appuiera pour la dernière sur le module `random`).
3. Écrire la fonction principale qui selon les options appelle la bonne fonction.

Exercice 3 :

Le but de cet exercice est de créer un script `code_propre` qui permet de faire des statistiques, de nettoyer ou de signaler des incorrections de syntaxes (pour une description des bonnes habitudes en python voir le site

`http://www.python.org/dev/peps/pep-0008/.`

On s'attachera surtout aux règles suivantes :
- nombre de caractère d'une ligne inférieur a 79 caractères.
- un espace autour des opérateurs binaires (affectation, test d'égalité...)
- un espace après une virgule mais pas avant

On pourra utiliser ce script dans un `shell` de la façon suivante :

```
code_propre -s <fichier>
     -> Calcule le nombre de ligne de codes vrai, le nombre de lignes
     blanches et le nombre de lignes de commentaires
code_propre -t <fichier>
     -> Remplace les tabulations par 4 espaces dans un fichier correct
code_propre -c <fichier>
     -> Signale signaler des infractions du code python aux regles de
     syntaxes
```

1. En utilisant le module `re`, écrire les fonctions permettant de faire des statistiques : `lignes_total` qui dénombre le nombre total de lignes, `lignes_vides` qui dénombre le nombre de lignes blanches, `lignes_comment` qui dénombre le nombre de lignes de commentaire, `lignes_doc` qui dénombre le nombre de lignes de documentation, `lignes_code` qui dénombre le nombre de lignes de code.
2. Écrire une fonction qui substitue quatre espaces aux tabulations qui apparaissent dans une chaîne de caractères.
3. Écrire les fonctions qui signalent les infractions aux règles de syntaxe d'une chaîne de caractère représentant une ligne de code : `test_long` qui teste si une chaîne de caractère est trop longue ; `test_ponct` qui signale les mauvaises utilisations de `,` et de `;` ; et `test_operateur` qui signale les opérateurs qui ne sont pas entourés par deux espaces. Écrire une fonction `signale` qui étant donné un numéro de ligne et une chaîne de caractère, appelle les différents tests et lorsqu'une infraction est découverte, imprime le numéro de la ligne, et le type d'infraction.
4. Écrire la fonction principale qui selon l'option appelle la bonne fonction avec les bons arguments.
5. Tester ce script sur les trois scripts que vous venez d'écrire.