LS4 - Travaux pratiques 7 bis

2 mai 2014

Objectifs : approfindir les expressions régulières.

1 Partie 1 : Les méthodes du module re

Nous avons vu comment utiliser les méthodes re.match et re.search pour trouver une occurrence d'un motif dans une chaîne. D'autres méthodes sont disponibles dans le module re lorsqu'on veut parcourir toutes les occurrences d'un motif :

`re.findall(motif,texte)` : retourne la liste de toutes les occurrences de motif trouvées. Si le motif contient k groupes (sous-expressions entre parenthèses), elle retourne une liste de k-uplets.

`re.finditer(motif, texte)` : retourne un itérateur qu'on peut parcourir avec for. A chaque itération il retourne un objet de la classe match qui correspond à la prochaine occurrence trouvée.

Finalement, on peut utiliser les méthodes `re.sub` pour faire des substitutions et `re.split` pour faire le découpage d'une chaîne suivant un motif.

Question 1

Une date suit l'un des formats suivants : 06/02/1978 ou 6-2-1978 06-02-1978 ou 6 février 1978. Ecrivez une expression régulière qui correspond à une date.

Ecrire une fonction `liste_dates(texte)` qui trouve toutes les dates dans un texte. Retourner les dates trouvées sous forme d'une liste de chaînes, dans l'ordre dans lequel elles apparaissent dans le texte. Utilisez la fonction `re.findall`.

q1 :

Question 2

Reprenez le même exercice, mais cette fois-ci, écrire une fonction `liste_dates_triplet(texte)` en utilisant les groupes (sous-expressions) pour retourner une liste de triplets (jour,mois,annee).

q2 :

Question 3

Reprenez le même exercice en utilisant re.finditer. Note : vous pouvez utiliser `re.finditer` à l'intérieur d'une compréhension de liste. Appelez la fonction `liste_dates_triplet_iter(texte)`.

q3 :

Question 4

Reprenez le même exercice, mais maintenant remplacez les mois numériques par leur nom (01-¿ janvier, 02 -¿ février,. . .). Utilisez la fonction re.finditer afin de parcourir la liste des résultats et effectuer la substitution. Appelez la fonction `liste_dates_triplet_nom_mois(texte)`

q4 :

La fonction `re.sub(motif, remplacer, chaine)` permet de remplacer les occurrences de motif par la chaine remplacer.

La fonction re.split(motif, chaine) scinde la chaine sur les occurrences du motif. Les morceaux sont retournés dans une liste.

Question 5

Ecrire une fonction `efface_voyelle(texte)` qui prend en entrée un mot et retourne le mot avec toutes les voyelles remplacées par des `_`. (Vous pouvez supposer que le texte est sans accents.)

q5 :

Question 6

Ecrire une fonction `decoupe_mots(texte)` qui prend en entrée un texte et retourne la liste des mots dans le texte. Le texte peut contenir de la ponctuation, des guillemets, des tabulations et des fins de ligne, des parenthèses. Essayez de traiter un maximum de cas. Testez votre fonction sur le texte suivant.

```
s="\tDans l'S, à une heure d'affluence. Un type dans les vingt-six ans, chapeau mou avec cordon re
```

q6 :

2 Partie 2 : Expressions régulières et dictionnaires

Nous allons réutiliser les fonctions suivantes :

Du TP5 : La fonction `afficher(canvas, T)` qui affiche le tableau T sur le canvas (ci-dessous).

Du TP7 : Les fonctions `extraire_ligne(l)` et `extraire_fichier(nomfichier)` qui extraient les données du fichier de statistiques sur les prénoms dans le fichier suivant :

Voir prenoms.txt. [1]

```
import tkinter

def afficher(canvas,T):
    canvas.delete("all")
    for i in range(len(T)):
        canvas.create_rectangle(20*i,400-T[i]*10,20*(i+1),400, fill="grey")
    canvas.update()

fenetre = tkinter.Tk()
quitter = tkinter.Button(fenetre, text="Quitter", command=fenetre.destroy)
quitter.pack()
canvas = tkinter.Canvas(fenetre,width=401,height=401,background="white")
canvas.pack()
afficher(canvas, [2,5,6,7,4,5,6])
```

Ecrivez une fonction `stat(canvas, d, motif)` qui prend en paramètre un canvas pour faire l'affichage, un dictionnaire {(nom,annee) :num} et un motif (chaine "raw") et qui affiche les statistiques par décennie pour les prénoms qui correspondent au motif. Par exemple, les prénoms terminant en "ine", les prénoms composés, les prénoms doubles, etc.

qpartie2 :

1. `Fichier fourni avec le sujet.`

3 Partie 3 : Script de nettoyage de code python

Le but de cet exercice est de nettoyer du code python, y compris l'indentation.

Question 7

Dans un code propre :
(R1) il n'y a aucun espace après une parenthèse ouvrante ou avant une parenthèse fermante.
(R2) il n'y aucun espace avant une virgule, deux points (:) ou point virgule (;).
(R3) il y a toujours un seul espace après une virgule, deux points (:) ou point virgule (;).
Ecrivez une fonction `diagnostic_ponctuation(code)` qui prend en entrée une chaîne multiligne de code et détecte les conditions ci-dessus. La fonction retourne None si aucune des conditions n'est détectée, et retourne une liste de paires (n,c) où n est le numéro de la ligne ou l'une de ces conditions n'est pas vérifiée, et un code c parmi (1,2,3) précisant quelle condition a été détectée.
q7 :

Question 8

Ecrivez une fonction `repare_ponctuation(ch)` qui met en conformité toute la chaîne selon les règles R1-R2-R3. La fonction retourne la chaîne modifiée.
q8 :

Question 9

(R4) Les lignes de code ne doivent pas dépasser 79 caractères.
Ecrire une fonction `diagnostic_long(ch)` qui détecte si une chaîne multiligne contient une ligne de plus de 79 caractères. La fonction retourne None si toutes les ligne sont de longueur au plus 79, sinon la fonction retourne le numéro de la première ligne où la longueur dépasse 79.
q9 :

Question 10

Pour couper les lignes de code trop longues il faut :
(1) Repérer les espaces en début de ligne (indentation)
(2) Trouver le dernier espace blanc parmi les 78 premiers caractères de la ligne
(3) Remplacer cet espace blanc par la chaîne " \\\n" suivi du même nombre de blancs en début de ligne plus 4.
Ecrire une fonction `repare_long(ch)` qui coupe ainsi toutes les lignes de longueur supérieure à 79.
q10 :

Question 11

(R5) Une ligne de code est bien indentée si elle commence avec un multiple de 4 espaces, et aucune tabulation (\t).
Ecrire une fonction `diagnostic_indentation(ch)` qui détecte s'il existe une ligne dans une chaîne multiligne ch qui ne commence pas par un multiple de 4 espaces. Si l'indentation est correcte, retourner None, sinon, retourner le numéro de la première ligne qui n'est pas correctement indentée.
q11 :

Question 12

Afin de rétablir une indentation correcte, nous allons procéder de la façon suivante.
Un compteur d'espaces est initialisé à 0.
Pour chaque ligne de code :
(1) Si cette ligne est plus indentée que la précédente, le compteur d'espaces est incrémenté de 4
(2) Si cette ligne est moins indentée que la ligne précédente, le compteur d'espaces est décrémenté de 4
Ecrire une fonction `repare_indentation(ch)` qui prend en entrée une chaîne multiligne et retourne une chaîne multiligne qui contient le même code, correctement indenté.
q12 :
Ecrivez un script `code_propre` qui s'utilise de la façon suivante à partir de la ligne de commande.

```
code_propre -p <fichier>
```

Execute la fonction diagnostic_ponctuation sur le code dans ¡fichier¿

```
code_propre -P <fichier1> <fichier2>
```

Execute la fonction repare_ponctuation sur le code dans ¡fichier1¿, écrit le résultat dans ¡fichier2¿

```
code_propre -l <fichier>
```

Execute la fonction diagnostic_long sur le code dans ¡fichier¿

```
code_propre -L <fichier1> <fichier2>
```

Execute la fonction repare_long sur le code dans ¡fichier1¿, écrit le résultat dans ¡fichier2¿

```
code_propre -i <fichier>
```

Execute la fonction diagnostic_indentation sur le code dans ¡fichier¿

```
code_propre -I <fichier1> <fichier2>
```

Execute la fonction repare_indentation sur le code dans ¡fichier1¿, écrit le résultat dans ¡fichier2¿

```
code_propre -a <fichier>
```

Effectue tous les diagnostics sur le code dans ¡fichier¿

```
code_propre -A <fichier1> <fichier2>
```

Effectue toutes les réparations sur le code dans ¡fichier1¿, écrit le résultat dans ¡fichier2¿
qcodepropre :