Langages de script (Python)
CMTP n° 5 : Expressions rationnelles

Dans ce sujet nous aborderons les expressions rationnelles (ou expressions régulières, (*regular expressions* en anglais, ou *regex*, ou *RE* en abrégé) en Python. Dans la suite on parlera de *motif*. Cela se fait avec le module `re`.
Pour commencer, on peut consulter :
https://docs.python.org/fr/3/howto/regex.html#regex-howto
On y trouvera la manière d'écrire des expressions des régulières en Python et comment les utiliser avec des chaînes de caractères. Il y a essentiellement deux approches : soit on compile la description du motif sous forme d'un *objet motif* sur lequel on peut appliquer des méthodes (comme `match(s)`, `search(s)` ou `fullmatch(s)`), soit on invoque une fonction du module `re` sur la description d'un motif et une chaîne (on dispose des fonctions `match(m,s)`, `search(m,s)` ou `fullmatch(m,s)`). Dans les deux cas, on utilise `match(-)` pour vérifier si le début d'une chaîne correspond à un motif, `search(-)` pour vérifier si une chaîne contient le motif donné, et enfin `fullmatch(s)` pour vérifier si une chaîne correspond à un motif.

Et pour la documentation complète du module, c'est :
https://docs.python.org/fr/3/library/re.html#module-re

Exercice 1 : Échauffement

Pour chaque motif ci-dessous, indiquer les chaînes correspondant *exactement* au motif, celles dont le début correspond au motif, celles contenant une sous-chaîne correspondant au motif et les autres...

Motif	Chaîne
`A.C`	`"abc"`
	`"AZZC"`
	`"AC"`
	`"ABC"`

Motif	Chaîne
`[ABC]`	`"A"`
	`"B"`
	`"AC"`
	`"ABC"`

Motif	Chaîne
`[a-k].*[1-5]`	`"a2"`
	`"A05"`
	`"ab12"`
	`"k1k"`

Motif	Chaîne
`[^a-k].*[1-5]`	`"a2"`
	`"A05"`
	`"ab12"`
	`"k1k"`

Motif	Chaîne
`[a-k].[1-5]{,3}`	`"ak1234"`
	`"a123"`
	`"ad12bd123"`
	`"z1k"`

Motif	Chaîne
`(a?b)+`	`"abababab"`
	`"aaaab"`
	`"bbbbbba"`
	`"bbbab"`

Motif	Chaîne
`(a|[AB]*)[0-9]`	`"aAB0"`
	`"AAB0"`
	`"0"`
	`"A-Z9"`

Motif	Chaîne
`.\^[^^]+.`	`a^0^0`
	`"aa^0^"`
	`"0^^^0"`
	`"A^Z9"`

Utiliser `fullmatch`, `match` et `search` pour vérifier vos réponses.
Attention : les expressions régulières utilisent la barre inverse `\` comme caractère spécial, comme par exemple dans le motif `.\^[^^]+.` ci-dessus. Ceci implique que dans la représentation d'un motif en tant que chaîne de caractères chaque `\` doit être doublé. Pour éviter la prolifération de barres obliques inverses vous pouvez utiliser les *chaînes brutes* de Python, en préfixant une chaîne avec `r` (pour *raw*, brut en anglais). Par exemple le motif `.\^[^^]+.` est représenté dans Python par la chaîne `".\\^[^^]+."` et par la chaîne brute `r".\^[^^]+."`.

Exercice 2 : filtrer

Dans cet exercice, on demande de définir des fonctions qui prennent en argument un nom de fichier et qui affichent les lignes de ce fichier qui respectent certaines propriétés.

1. `filtre1` : afficher les lignes qui commencent par un mot d'au moins 8 et d'au plus 12 caractères alphanumériques.
2. `filtre2` : afficher les lignes qui contiennent le **mot** `toto`.
3. `filtre3` : afficher les lignes qui contiennent au moins trois fois le **mot** `toto`.
4. `filtre4` : afficher les lignes qui contiennent au moins trois fois le même **mot** sur l'alphabet `[a-z]`.

Exercice 3 : Les prénoms en France 1

Pour cet exercice et le prochain, nous utiliserons le fichier `prenoms.txt` que vous trouverez sur Moodle. Ce fichier provient des données *open data* sur les prénoms en France du 1905 au 2015, disponible également sur le site http://www.data.gouv.fr/. Comme le fichier est assez gros, veillez à optimiser le temps de réponse de vos programmes pour que celui-ci soit raisonnable.
Chaque ligne de `prenoms.txt` contient une chaîne de la forme : `annee,nombre,idn,Prenom` (sans espace après les virgules), où :

- `annee` est un nombre de 4 chiffres donnant une année,
- `nombre` est le nombre des nouveaux nés en `annee` ayant eu le prénom `Prenom`,
- `idn` est un entier donnant un identificateur numérique à `Prenom`,
- `Prenom` est une chaîne quelconque non vide.

1. Écrire une fonction `parse_line(s)` qui prend en entrée une chaîne `s` de la forme `annee,nombre,idn,Prenom` et retourne un 4-uplet (de 3 entiers et une chaîne) de la forme `(annee, nombre, idn, Prenom)`. Pour cela, on utilisera la fonction `match` et les références aux groupes délimités par des parenthèses dans une expression rationnelle.
2. Écrire une fonction `parse_file(path)` qui prend en entrée un nom de fichier texte et qui, pour l'ensemble des lignes du fichier qui suivent le format de la question précédente, retourne un dictionnaire associant à chaque couple `(Prenom,annee)` le nombre de fois où le prénom a été choisi pendant cette année.

   En plus, la fonction doit afficher les lignes du fichier qui ne correspondent pas au format décrit ci-dessus (il devrait y en avoir 6 dans le fichier `prenoms.txt`).

   Attention : le fichier `prenoms.txt` peut contenir plusieurs lignes avec exactement le même couple `(Prenom,annee)` !
3. Écrire une fonction `name_frequency(d)` qui prend en entrée un dictionnaire dont les clefs sont des paires `(Prenom,annee)` et les valeurs sont les nombres de fois les prénoms ont été choisis cette année, et retourne un dictionnaire dont les clefs sont les prénoms et la valeur associée est le nombre total de fois où le prénom a été choisi (sur toutes les années).
4. Écrire une fonction `popular_name(fname,n)` qui prend en entrée un nom de fichier `fname` contenant une liste des prénoms sous le format décrit ci-dessus et un entier `n` et affiche les `n` prénoms les plus utilisés selon le fichier.

   Par exemple sur le fichier `prenoms.txt` et l'entier `5` la fonction devrait afficher :

```
Marie 2290210
Jean 1962089
Pierre 885734
Michel 818404
Andre 710477
```

Exercice 4 : Les prénoms en France 2

Écrire un script pour répondre aux questions suivantes qui portent sur les données du fichier `prenoms.txt` :

1. Combien de prénoms sont des carrés (c'est à dire des chaînes de caractères de la forme `rr` pour une certaine chaîne `r`) ? La réponse devrait être 18.

   (*Astuce : écrire une fonction auxiliaire* `is_square` *en utilisant les groupes sur les expressions régulières. Attention : la première lettre des prénoms dans* `prenoms.txt` *est une majuscule.*)
2. Combien de prénoms contiennent un carré de longueur au minimum 4 ? La réponse devrait être 63.
3. Combien de prénoms contiennent au moins 5 fois la même lettre ? La réponse devrait être 3.
4. Combien de prénoms contiennent au moins 4 voyelles consécutives ? La réponse devrait être 25.

   (*Astuce : utiliser le modificateur* `{m,n}` *qui permet de spécifier entre entre* `m` *et* `n` *répétitions de l'expression qui précède.*)
5. Combien de prénoms contiennent 4 consonnes consécutives ? La réponse devrait être 7.

   (*Attention aux apostrophes, espaces et traits d'union dans le prénoms*).
6. Combien de prénoms finissent par 4 consonnes consécutives ? La réponse devrait être 1.

7. Quelle est la proportion de prénoms composés (contenant un espace ou un trait d'union) ? La réponse devrait être 0.077 (arrondie au millième).
8. Quelle proportion de la *population* née entre 1905 et 2015 a un prénom composé ? La réponse devrait être 0.030 (arrondi au millième).
9. Pour chaque lettre, quel est le prénom le plus populaire qui commence par cette lettre ?

Exercice 5 : Star Wars

Nous voulons récupérer le script de l'Episode IV de Star Wars : "*A new Hope*". Nous avons trouvé une page web `Star-Wars-A-New-Hope.html` contenant le texte du script et nous voulons l'extraire dans un format textuel plus approprié.
Remarque : Nous utilisons ici les fonctionnalités du module `re`, notamment les méthodes `re.search` et `re.sub`, mais en général il y a des modules dédiées au traitement du `html` en Python.

1. Télécharger depuis Moodle le fichier `Star-Wars-A-New-Hope.html`, ouvrir le code source avec un éditeur de texte. Quelles sont les balises qui délimitent le texte du script du film ?
2. Écrire un script qui crée un nouveau fichier texte ne contenant que le script du film et éventuellement les balises `<b>` et `</b>`.
   (*Astuce : l'option `DOTALL` fait en sorte que `.` corresponde à n'importe quel caractère (y compris le caractère de retour à la ligne). Par exemple `re.search("0(.*)0", s, re.DOTALL)` cherche une chaîne de caractères dans `s` délimitée par le chiffre `0` et contenant éventuellement des retours à la ligne.*)
3. Quels sont les rôles des chaînes de caractères délimitées par la balise `<b>` ? Comment reconnaître le nom d'un personnage qui commence un dialogue par rapport aux autres chaînes délimités par `<b>` ? Comment reconnaître la fin d'un dialogue d'un personnage ?
   Modifier le script afin de générer un fichier texte tel que :
   - un dialogue soit introduit par le nom du personnage parlant en majuscule suivit par deux-points et ensuite le dialogue délimité par les guillemets, comme par exemple :
     `BEN'S VOICE: "Luke, the Force will be with you."`
   - aucune balise `<b>`, `</b>` n'y apparaît plus ;
   - tout le texte doit être aligné à gauche, aucune nouvelle ligne ne peut commencer avec des espaces vides, et il ne doit plus y avoir plusieurs espaces vides consécutifs.

   (*Astuce : les répétitions comme `*` sont gloutonnes (ils cherchent la chaîne la plus longue compatible avec le motif), pour modifier ce comportement et chercher les sous-chaîne compatibles les plus courtes possibles, ajouter le modificateur `?`, comme dans `*?`*).