Expressions régulières

Nous vous proposons dans ce notebook quelques exercices sur les expressions régulières. Faisons quelques remarques avant de commencer :

- nous nous concentrons sur l'écriture de l'expression régulière en elle-même, et pas sur l'utilisation de la librairie ;
- en particulier, tous les exercices font appel à `re.match` entre votre ***regexp*** et une liste de chaînes d'entrée qui servent de jeux de test - ce qui signifie notamment que vous pouvez ou pas commencer votre regexp par le caractère `^`, cela ne changera rien au résultat de la correction, puisqu'en utilisant `match` on impose que la correspondance doit avoir lieu au debut de la chaîne.

Liens utiles

Pour travailler sur ces exercices, vous pouvez profitablement avoir sous la main :

- la [documentation officielle](https://docs.python.org/2/library/re.html (https://docs.python.org/2/library/re.html)

regular-expression-syntax),

- et cet outil interactif sur regexp101.com (http://regex101.com/#python) qui permet d'avoir un retour presque immédiat, et donc d'accélérer la mise au point.

Exercice - niveau basique

Identificateurs python

Écrivez une expression régulière qui décrit ce qu'on peut utiliser comme nom de variable en python.

```
# quelques exemples
from corrections.w6_regexp import exo_pythonid
exo_pythonid.exemple()


regexp_pythonid = "<votre_code>"


# pour valider votre code
exo_pythonid.correction(regexp_pythonid)
```

Exercice - niveau intermédiaire

Méthodes spéciales

Nous avons vu la semaine passée les méthodes spéciales utilisées par python pour vous permettre de redéfinir le comportement du langage avec les objets de vos classes.

Le but de cet exercice est d'écrire une expression régulière qui décrive une classe de noms de méthodes, susceptibles d'être considérés comme des noms de méthodes spéciales.

Pour préciser l'exercice, on recherche des noms de méthodes qui doivent :

1. commencer par exactement deux underscores (appelés également tirets-bas) `_`,
2. se terminer par exactement deux underscores,
3. contenir dans la partie centrale un identificateur qui pourrait être un identificateur python (mais qui pour respecter les deux premiers points ne peut pas commencer ou finir par un underscore)

```
# quelques exemples
from corrections.w6_regexp import exo_specials
exo_specials.exemple()


regexp_specials = "<votre_regexp>"


# validation
exo_specials.correction(regexp_specials)
```

Exercice - niveau avancé

Vu comment sont conçus les exercices, vous ne pouvez pas passer à `re.compile` un ***flag*** comme `re.IGNORECASE` ou autre ; sachez cependant que vous pouvez ***embarquer* ces flags dans la *regexp*** elle-même ; par exemple pour rendre la regexp insensible à la casse de caractères, au lieu d'appeler `re.compile` avec le flag `re.I`, vous pouvez utiliser `(?i)` comme ceci :

```
import re
re.match("(?i)abc","ABC").group(0)
```

Remarquez bien que l'emplacement où vous ajoutez le ***flag*** dans la ***regexp*** n'a pas d'importance en général (lire la doc pour le flag `x`), le flag agit toujours sur toute la ***regexp***.

```
re.match("abc(?i)","ABC").group(0)
```

Pour plus de précisions sur ce trait, que nous avons laissé de coté dans le complément pour ne pas trop l'alourdir, voyez la documentation sur les expressions régulières (https://docs.python.org/2/library/re.html#regular- expression-syntax) et cherchez la première occurrence de `iLmsux`.

Décortiquer une URL

On vous demande d'écrire une expression régulière qui permette d'analyser des URLs.

Voici les conventions que nous avons adoptées pour l'exercice :

- la chaîne contient les parties suivantes
  - `<protocol>://<location>/<path>`
- l'url commence par le nom d'un protocole qui doit être parmi `http`, `https`, `ftp`, `ssh`
- le nom du protocole peut contenir de manière indifférente des minuscules ou des majuscules,
- ensuite doit venir la séquence `://`
- ensuite on va trouver une chaîne `<location>` qui contient :
  - potentiellement un nom d'utilisateur, et s'il est présent, potentiellement un mot de passe,
  - obligatoirement un nom de `hostname`,
  - potentiellement un numéro de port;
- lorsque les 4 parties sont présentes dans `<location>`, cela se présente comme ceci :
  - `<location> = <user>:<password>@<hostname>:<port>`
- si l'on note entre crochets les parties optionnelles, cela donne :
  - `<location> = [<user>[:<password>]@]<hostname>[:<port>]`
- le champ `<user>` ne peut contenir que des caractères alphanumériques; si le `@` est présent le champ `<user>` ne peut pas être vide
- le champ `<password>` peut contenir tout sauf un `:` et de même, si le `:` est présent le champ `<password>` ne peut pas être vide
- le champ `<hostname>` peut contenir un suite non-vide de caractères alphanumériques, underscores, ou `.`
- le champ `<port>` ne contient que des chiffres, et il est non vide si le `:` est spécifié
- le champ `<path>` peut être vide.

Enfin, vous devez définir les groupes `proto`, `user`, `password`, `hostname`, `port` et `path` qui sont utilisés pour vérifier votre résultat. Dans la case `Résultat attendu`, vous trouverez soit `None` si la regexp ne filtre pas l'intégralité de l'entrée, ou bien une liste ordonnée de tuples qui donnent la valeur de ces groupes ; vous n'avez rien à faire pour construire ces tuples, c'est l'exercice qui s'en occupe.

```
from corrections.w6_regexp import exo_url
```

```
exo_url.exemple()
```

```
# n'hésitez pas à construire votre regexp petit à petit

regexp_url = "<votre_regexp>"
```

```
exo_url.correction(regexp_url)
```