Jour 18 Expressions régulières

Expressions régulières

Une expression régulière ou un regex est une chaîne de texte spéciale qui aide à trouver des mo dèles dans les données. Un regex peut être utilisé pour vérifier si un modèle existe dans un type de données différent. Pour utiliser Regex dans Python en premier, nous devons importer le mo dule Regex qui est appelé *re*.

Le module ***re***

Après l'importation du module, nous pouvons l'utiliser pour détecter ou trouver des modèles.

```
Importer RE
```

Méthodes dans le module ***re***

Pour trouver un modèle, nous utilisons différents ensembles de caractères *re* qui permet de r echercher une correspondance dans une chaîne.

- *re.match()*: ne recherche qu'au début de la première ligne de la chaîne et renvoie des objets correspondants si trouvés, sinon ne renvoie aucun.
- *re.search*: Renvoie un ob jet de correspondance s'il y en a un n'importe où dans la chaîne, y compris les chaînes multilines.
- *re.findall*: renvoie une liste contenant toutes les correspondances

- *re.split*: prend une chaîne, la divise aux points de match, renvoie une liste
- *re.sub*: remplace une ou plusieurs correspondances dans une chaîne

Correspondre

```
# syntac re.match (substring, string, re.i) # substring est une chaîne ou un motif, la chaîne est le texte que nous recherchons pour un motif, re.i est un cas ignore
```

```
Importer RE

txt = 'J'adore enseigner Python et JavaScript' # Il renvoie un objet avec Span, et correspond à Match = re.match ('I Love To Teach', txt, re.i) print (match) # < re.match Object; Span = ( 0, 15 ), match = 'j'aime enseigner' > # Nous pouvons obtenir la position de démarrage et de fin du match en tant que tuple en utilisant Span Span = match.span () print (span) # (0, 15) # Lets Find the Start and Stop Position de Span Start, fin = Span = txt [start: end]
```

```
print(substring)        # I love to teach
```

Comme vous pouvez le voir dans l'exemple ci-dessus, le modèle que nous recherchons (ou la sous-chaîne que nous recherchons) est *I love to teach*. La fonction de correspon dance renvoie un objet uniquement si le texte commence par le motif.

```
Importer RE

txt = 'J'adore enseigner à Python et JavaScript' Match = re.match ('I L
ike to Teach', txt, re.i)
imprimer (match) # aucun
```

La chaîne ne chaîne pas avec *I like to teach*, donc il n'y avait pas de correspondance et la m éthode de correspondance n'a renvoyé aucune.

Recherche

```
# syntaxe
re.match (substring, string, re.i) # substring est un motif, la chaîne est le texte que nous re
cherchons pour un motif, re.i est un cas
```

```
Importer RE

txt = " 'Python est le plus beau langage qu'un être humain ait jamais créé.

Je recommande Python pour un premier langage de programmation " '

# Il renvoie un objet avec Span et Match Match = Re.Search ('First', txt, re.i) print (match) #
< re.match objet; Span = ( 100, 105 ), match = 'First' > # Nous pouvons obtenir la position d
e démarrage et de fin du match en tant que tuple en utilisant Span Span = match.span () print
(span) # (100, 105) # Lets Find the Start and Stop Position From the Span Start, fin = Span I
mprime = txt [start: end] print (substring) # d'abord
```

Comme vous pouvez le voir, la recherche est bien meilleure que la correspondance car elle peu t rechercher le motif tout au long du texte. La recherche renvoie un objet de match avec un pre mier match trouvé, sinon il renvoie *None*. Une fonction bien meilleure *re* est *findall*. Cette fon ction vérifie le modèle via toute la chaîne et renvoie toutes les correspondances en tant que list e.

Recherche de toutes les correspondances à l'aide de ***findall***

*findall()*Renvoie tous les matchs en tant que liste

```
txt = " 'Python est le plus beau langage qu'un être humain ait jamais créé.  Je recommande Python pour un premier langage de programmation " '


# Il renvoie une liste correspond à = re.findall ('linguisse', txt, re.
i)
imprimer (matchs) # ['Langue', 'Langue']
```

Comme vous pouvez le voir, le mot *language* a été trouvé deux fois dans la chaîne. Laissez-n ous pratiquer un peu plus. Maintenant, nous allons chercher des mots python et python dans la chaîne:

```
txt = " 'Python est le plus beau langage qu'un être humain ait jamais créé.  Je recommande Python pour un premier langage de programmation " '


# Il renvoie la liste correspond = re.findall ('python', txt, re.i)

print (correspond) # ['python', 'python']
```

Puisque nous utilisons *re.I*, les lettres minuscules et majuscules sont incluses. Si nous n'avons pas le drapeau re.i, alors nous devrons écrire notre modèle différemment. Laissez-nous vérifier :

```
txt = " 'Python est le plus beau langage qu'un être humain ait jamais créé.

Je recommande Python pour un premier langage de programmation " '

correspond à = re.findall ('python | python', txt)
print (correspond) # ['python', 'python']

# correspond = re.findall ('[pp] ython', txt) print (correspon
d) # ['python', 'python']
```

Remplacement d'une sous-chaîne

```
txt = " 'Python est le plus beau langage qu'un être humain ait jamais créé.

Je recommande Python pour un premier langage de programmation " '

Match_Replaced = re.sub ('python | python', 'javascript', txt, re.i)

print (match_replacen) # javascript est le plus beau langage qu'un être humain ait jamais créé.
# Ou match_replaced = re.sub ('[pp] ython', 'javascript', txt, re.i) imprimer (match_replaced) #
javascript est le plus beau langage qu'un être humain ait jamais créé.
```

Ajoutons un autre exemple. La chaîne suivante est vraiment difficile à lire à moins que nous ne supprimons le symbole%. Le remplacement du% par une chaîne vide nettoiera le texte.

```
txt = " '% i% m te %% a %% che% r% a% n% d %% i l% o% ve te% gch% ing.  T% he% r e i% s n% o% th% ing en tant que R% en éveillant un% s e% duc% à% i% ng a% n% d e% m% p% ow% er% in% e% o% ple.
I Fo% und te% a% ching m% minerai i% n% t% er %% es% ting t% h% an autre% emplois.

D% o% es thi% s m% ot% iv% a% te% y% o% u à b% e a t% e% a% cher? " '

correspond à = re.sub ('%', '', txt)
Imprimer (matchs)
```

```
Je suis professeur et j'aime enseigner.
Il n'y a rien d'aussi gratifiant que l'éducation et l'autonomisation des gens.

J'ai trouvé l'enseignement plus intéressant que tous les autres emplois. Cela vous motive-t-il à être enseignant?
```

Fissure du texte à l'aide de l'expression regex

```
txt = " 'Je suis professeur et j'adore enseigner.  Il n'y a rien d'aussi gratifiant que l'éducation et l'autonomisation des gens.  J'ai trouvé l'enseignement plus intéressant que tous les autres empl ois.  Cela vous motive-t-il à être enseignant? " 'Imprimer (re.split (' \ n ', txt)) # Splating Utilis ation de \ n - Symbole de fin de ligne
```

```
[«Je suis enseignant et j'aime enseigner.», «Il n'y a rien d'aussi gratifiant que d'éduquer et d 'autonomiser les gens.
```

Écrire des modèles regex

Pour déclarer une variable de chaîne, nous utilisons un devis simple ou double. Pour déclarer l a variable regex ***r''***. Le modèle suivant identifie uniquement Apple avec des minuscules, pour r endre son cas insensible, soit nous devons réécrire notre modèle, soit nous devons ajouter un dr apeau.

```
Importer RE

regex_pattern = r'apple 'txt =' Apple et Banana sont des fruits. Un vieux cliché dit qu'une po mme par jour où un médecin a été remplacée par une banane par jour maintient le médecin l oin. 'correspond à = re.findall (regex_pattern, txt) print (correspond) # [' Apple ']
```

```
# Pour faire des correspondances insensibles à l'ajout insensible 'Matches = re.findall (r
egex_pattern, txt, re.i) imprimer (correspond) # [' apple ',' Apple '] # ou nous pouvons ut
iliser un ensemble de caractères regex_pattern = r' [aa] pple '# Cela signifie que la prem
ière lettre pourrait être Apple ou Apple Matches = re.findall (regex_pattern, txt) print (c
orrespond) # ['Apple', 'Apple']
```

- []: Un ensemble de caractères o [a-c] signifie, a ou b ou c o [a-z] signifie que toute lettr e de a à z o [a-z] signifie tout caractère de a à z o [0-3], 0 ou 1 ou 2 ou 3 o [0-9] signifie tout nombre de 0 à 9 o [a-za-z0-9] \ d signifie: correspondez à l'endroit où la chaîne cont ient des chiffres (nombres de 0 à 9) o \ d signifie: correspondez à l'endroit où la chaîne n e contient pas de chiffres •. : tout caractère sauf le nouveau caractère de ligne (\ n)

- ^ commence par o r '^substring' eg r '^love', une phrase qui commence par un mot love o r '[^ABC] signifie pas a, pas b, pas c.

- $: se termine par o r'Substring $ 'par exemple r'love $', phrase qui se termine p ar un mot love • *: zéro ou plus de fois o r '[a] *' signifie facultatif ou il peut se produire plusieurs fois.

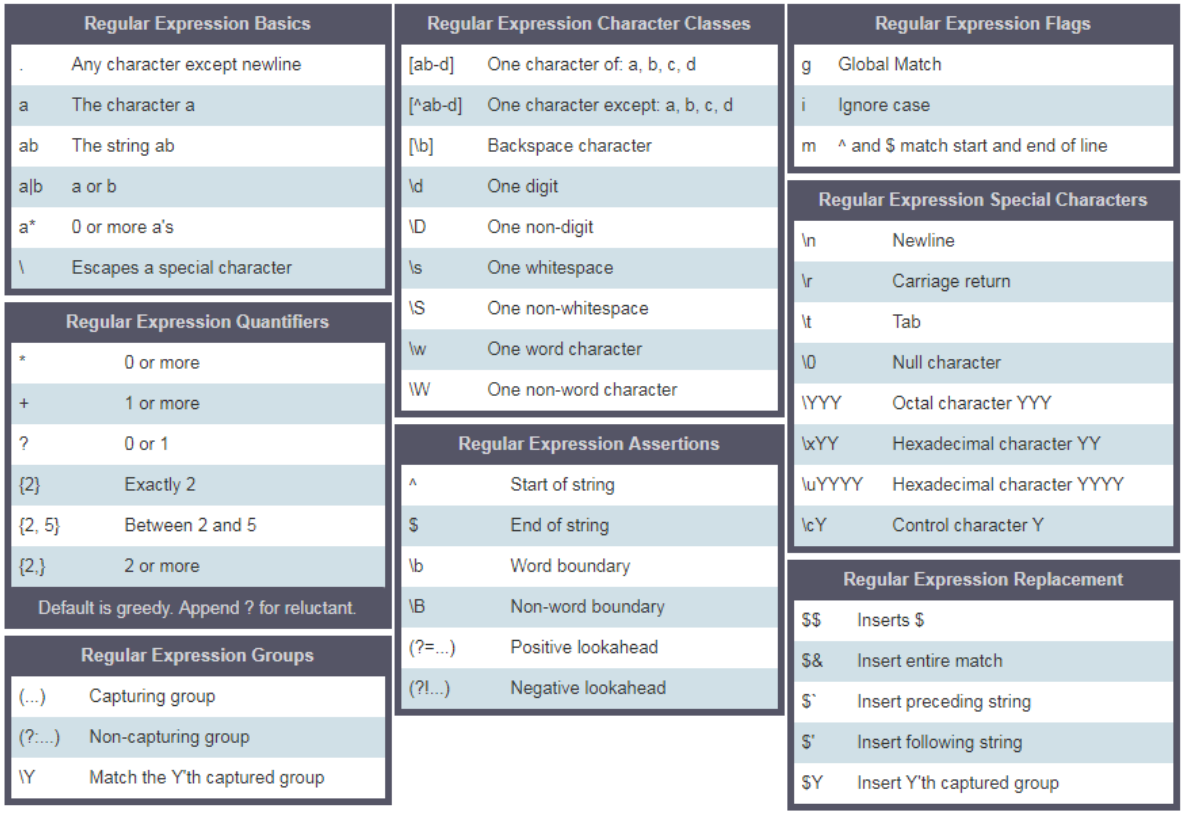
- +: une ou plusieurs fois o r '[a] +' signifie au moins une fois (ou plus)

- ?: zéro ou une fois o r '[a]?' signifie zéro fois o u une fois • {3}: exactement 3 caractères • {3,} : au moins 3 caractères

- {3,8}: 3 à 8 caractères
- |: l'un ou l'autre ou

  o r'apple | banane 'signifie pomme ou banane

- (): Capture et groupe



Regular Expression Basics	
.	Any character except newline
a	The character a
ab	The string ab
a|b	a or b
a*	0 or more a's
\	Escapes a special character

Regular Expression Quantifiers	
*	0 or more
+	1 or more
?	0 or 1
{2}	Exactly 2
{2, 5}	Between 2 and 5
{2,}	2 or more

Default is greedy. Append ? for reluctant.

Regular Expression Groups	
(...)	Capturing group
(?:...)	Non-capturing group
\Y	Match the Y'th captured group

Regular Expression Character Classes	
[ab-d]	One character of: a, b, c, d
[^ab-d]	One character except: a, b, c, d
[\b]	Backspace character
\d	One digit
\D	One non-digit
\s	One whitespace
\S	One non-whitespace
\w	One word character
\W	One non-word character

Regular Expression Assertions	
^	Start of string
$	End of string
\b	Word boundary
\B	Non-word boundary
(?=...)	Positive lookahead
(?!...)	Negative lookahead

Regular Expression Flags	
g	Global Match
i	Ignore case
m	^ and $ match start and end of line

Regular Expression Special Characters	
\n	Newline
\r	Carriage return
\t	Tab
\0	Null character
\YYY	Octal character YYY
\xYY	Hexadecimal character YY
\uYYYY	Hexadecimal character YYYY
\cY	Control character Y

Regular Expression Replacement	
$$	Inserts $
$&	Insert entire match
$`	Insert preceding string
$'	Insert following string
$Y	Insert Y'th captured group

Utilisons des exemples pour clarifier les caractères méta-ci-dessus

Crochet

Laissez-nous utiliser le crochet pour inclure le bas et le haut du boîtier

```
regex_pattern = r '[aa] pple' # Ce support carré signifie un ou un txt = 'la pomme et la banane sont des fruits. Un vieux cliché dit qu'une pomme par jour où un médecin a été remplacée par une banane par jour maintient le médecin loin.



correspond à = re.findall (regex_pattern, txt)
Imprimer (matchs) # ['Apple', 'Apple']
```

Si nous voulons rechercher la banane, nous écrivons le modèle comme suit:

```
regex_pattern = r '[aa] pple | [bb] anana' # Ce support carré signifie un ou un txt = 'la pom me et la banane sont des fruits. Un vieux cliché dit une pomme par jour où un médecin a ét é remplacé par une banane par jour

maintient le médecin loin.  correspond à = re.findall (regex_p
attern, txt)
```

```
Imprimer (matchs) # [«pomme», «banane», «pomme», «banane»]
```

À l'aide du crochet et / ou de l'opérateur, nous parvenons à extraire la pomme, la pomm
e, la banane et la banane.

Caractère d'échappement (\) dans Regex

```
regex_pattern = r '\ d' # d est un caractère spécial qui signifie des chiffres

txt = 'Cet exemple d'expression régulière a été fait le 6 décembre
2019 et révisé le 8 juillet 2021 '
correspond à = re.findall (regex_pattern, txt) print (correspond) # ['6', '2', '0', '1', '9', '8', '2'
, '0', '2', '1'], ce n'est pas ce que nous voulons
```

Une ou plusieurs fois (**+**)

```
regex_pattern = r '\ d +' # d est un caractère spécial qui signifie chiffres, + signifie une ou plusi
eurs fois
txt = 'Cet exemple d'expression régulière a été fait le 6 décembre
2019 et révisé le 8 juillet 2021 '
correspond à = re.findall (regex_pattern, txt) print (correspond) # ['6', '2019', '8', '2021'] -
Maintenant, c'est mieux!
```

Période(.)

```
regex_pattern = r '[a].'  # Ce support carré signifie a et.  signifie tout caractère sauf la nouvel
le ligne txt = " 'Apple et Banana sont des fruits' " correspond aux correspondances = re.finda
ll (regex_pattern, txt) print (correspond) # ['an', 'an', 'an', 'a', 'ar']



regex_pattern = r '[a]. +' #. Tout caractère, + tout caractère un ou plusieurs fois correspond à
= re.findall (regex_pattern, txt) print (correspond) # ['et banane sont des fruits']
```

Zéro ou plus de fois (*)

Zéro ou plusieurs fois. Le modèle peut ne pas se produire ou il peut se produire plusieurs fois.

```
regex_pattern = r '[a]. *' #. Tout caractère, * tout caractère zéro ou plus de fois txt = " 'Apple
et la banane sont des fruits' "
```

```
correspond à = re.findall (regex_pattern, txt) print (correspond) # [
'et banane sont des fruits']]
```

Zéro ou une fois (?)

Zéro ou une fois. Le motif peut ne pas se produire ou il peut se produire une fois.

```
txt = " 'Je ne sais pas s'il existe une convention comment écrire le mot e-mail.  Certaines perso
nnes l'écrivent comme e-mail, d'autres peuvent l'écrire comme e-mail ou e-mail. " 'Regex_patt
ern = r' [ee] -? Mail '#? signifie ici que '-' est des matchs facultatifs = re.findall (regex_pattern
, txt) print (correspond) # ['e-mail', 'e-mail', 'e-mail', 'e-mail']
```

Quantifier en regex

Nous pouvons spécifier la longueur de la sous-chaîne que nous recherchons dans un texte, en utilisant un support bouclé. Imaginons, nous sommes intéressés par une sous-chaîne avec un e longueur de 4 caractères:

```
txt = 'Cet exemple d'expression régulière a été fait le 6 décembre
2019 et révisé le 8 juillet 2021 'regex_pattern = r' \ d {4} '# Exactemen
t quatre fois correspond à = re.findall (regex_pattern, txt) print (corresp
ond) # [' 2019 ',' 2021 ']


txt = 'Cet exemple d'expression régulière a été fait le 6 décembre
2019 et révisé le 8 juillet 2021 'regex_pattern = r' \ d {1, 4} '# 1 à 4
matchs = re.findall (regex_pattern, txt) print (correspond) # [' 6 ',' 2
019 ',' 8 ',' 2021 ']
```

Cart **^**

- Commence par

```
txt = 'Cet exemple d'expression régulière a été fait le 6 décembre
2019 et révisé le 8 juillet 2021 '
```

```
regex_pattern = r '^Cet' # ^signifie commence par des correspondances =
re.findall (regex_pattern, txt) print (correspond) # ['this']
```

- Négation

```
txt = 'Cet exemple d'expression régulière a été fait le 6 décembre
2019 et révisé le 8 juillet 2021 'regex_pattern = r' [^a-za-z] + '# ^dans le caractère set signi
fie négation, pas a to z, pas a to z, pas d'espace correspond = '8', '2021']
```

Exercices: Jour 18

Exercices: niveau 1

1. Qu'est-ce le que mot le plus fréquent dans le paragraphe suivant APH?

```
Paragraphe = 'J'adore enseigner. Si vous n'aimez pas enseigner ce que vous pouvez aimer d'au
tre. J'adore Python si vous n'aimez pas quelque chose qui peut vous donner toutes les capacité
s de développer une application que pouvez-vous aimer d'autre.
```

```
[
(6, «amour»),
(5, «vous»),
(3, «peut»),
(2, «quoi»),
(2, «enseignement»),
(2, «pas»),
(2, «else»),
(2, «faire»),
(2, «je»),
(1, «qui»),
(1, «à»),
(1, «le»),
(1, «quelque chose»),
(1, «si»),
(1, «donner»),
(1, «développer»),
(1, «capacités»),
(1, «application»),
(1, «an»),
(1, «tout»),
(1, «python»),
(1, «si»)
]]
```

2. La position de certaines particules sur l'axe X horizontal est -12, -4, -3 et -1 dans le sens négatif, 0 à l'origine, 4 et 8 dans le sens positif. Extraire ces nombres de tout ce texte et tro uver la distance entre les deux particules les plus éloignées.

```
Points = ['-12', '-4', '-3', '-1', '0', '4', '8'] trid_points = [-12, -4, -3, -1, -1, 0, 2, 4
, 8] Distance = 8 - (- 12) # 20 # 20
```

Exercices: niveau 2

1. Écrivez un modèle qui s'identifie si une chaîne est une variable Python valide

```
is_valid_variable ('first_name') # true is_valid_variable ('prem
ier-nom') # false is_valid_variable ('1First_name') # false is_v
alid_variable ('firstname') # true '
```

Exercices: niveau 3

1. Nettoyez le texte suivant. Après le nettoyage, comptez trois mots les plus fréquents de la chaîne.

```
phrase = " '% i $ am @% a% tea @ cher%, & & i lo% # ve% tea @ ching%;. Il n'y a rien
; & as & mo @ re-récompense comme educa @ ting & & & @ emp% o @ wering peo
@ ple. ; J'ai trouvé le thé @ ching m% o @ re intéressant tha @ n tout autre% jo @ bs.
% Do @ es thi% s mo @ tivate yo @ u pour être un thé @ cher !? " '



print (clean_text (phrase));
Je suis enseignant et j'aime enseigner qu'il n'y a rien d'aussi gratifiant que d'éduquer et d'autono
miser les gens que j'ai trouvés plus intéressants que tout autre emploi, cela vous motive à être u
n enseignant



print (Most_Frequent_Words (Cleaned_Text)) # [(3, «i»), (2, «enseignement»), (2, «pro
fesseur»)]
```

Félicitations!