TD2

January 21, 2021

1 TD 2 : manipuler des expressions régulières avec Python

1/ Quel est le package python qui permet de gérer les expression régulières ? importez-le et affichez l'aide correspondante avec la fonction help()

[1]:
```
#importez le package
import re
#affichez l'aide
help(re)
```

```
Help on module re:

NAME
    re - Support for regular expressions (RE).

MODULE REFERENCE
    https://docs.python.org/3.6/library/re

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module provides regular expression matching operations similar to
    those found in Perl.  It supports both 8-bit and Unicode strings; both
    the pattern and the strings being processed can contain null bytes and
    characters outside the US ASCII range.

    Regular expressions can contain both special and ordinary characters.
    Most ordinary characters, like "A", "a", or "0", are the simplest
    regular expressions; they simply match themselves.  You can
    concatenate ordinary characters, so last matches the string 'last'.

    The special characters are:
        "."      Matches any character except a newline.
        "^"      Matches the start of the string.
```

```
        "$"      Matches the end of the string or just before the newline at
                 the end of the string.
        "*"      Matches 0 or more (greedy) repetitions of the preceding RE.
                 Greedy means that it will match as many repetitions as
possible.
        "+"      Matches 1 or more (greedy) repetitions of the preceding RE.
        "?"      Matches 0 or 1 (greedy) of the preceding RE.
        *?,+?,?? Non-greedy versions of the previous three special characters.
        {m,n}    Matches from m to n repetitions of the preceding RE.
        {m,n}?   Non-greedy version of the above.
        "\\"     Either escapes special characters or signals a special
sequence.
        []       Indicates a set of characters.
                 A "^" as the first character indicates a complementing set.
        "|"      A|B, creates an RE that will match either A or B.
        (…)    Matches the RE inside the parentheses.
                 The contents can be retrieved or matched later in the string.
        (?aiLmsux) Set the A, I, L, M, S, U, or X flag for the RE (see below).
        (?:…)  Non-grouping version of regular parentheses.
        (?P<name>…) The substring matched by the group is accessible by name.
        (?P=name)     Matches the text matched earlier by the group named name.
        (?#…)  A comment; ignored.
        (?=…)  Matches if … matches next, but doesn't consume the string.
        (?!…)  Matches if … doesn't match next.
        (?<=…) Matches if preceded by … (must be fixed length).
        (?<!…) Matches if not preceded by … (must be fixed length).
        (?(id/name)yes|no) Matches yes pattern if the group with id/name
matched,
                           the (optional) no pattern otherwise.

    The special sequences consist of "\\" and a character from the list
    below.  If the ordinary character is not on the list, then the
    resulting RE will match the second character.
        \number  Matches the contents of the group of the same number.
        \A       Matches only at the start of the string.
        \Z       Matches only at the end of the string.
        \b       Matches the empty string, but only at the start or end of a
word.
        \B       Matches the empty string, but not at the start or end of a
word.
        \d       Matches any decimal digit; equivalent to the set [0-9] in
                 bytes patterns or string patterns with the ASCII flag.
                 In string patterns without the ASCII flag, it will match the
whole
                 range of Unicode digits.
        \D       Matches any non-digit character; equivalent to [^\d].
        \s       Matches any whitespace character; equivalent to [ \t\n\r\f\v]
in
```

```
                bytes patterns or string patterns with the ASCII flag.
                In string patterns without the ASCII flag, it will match the
whole
                range of Unicode whitespace characters.
        \S      Matches any non-whitespace character; equivalent to [^\s].
        \w      Matches any alphanumeric character; equivalent to [a-zA-Z0-9_]
                in bytes patterns or string patterns with the ASCII flag.
                In string patterns without the ASCII flag, it will match the
                range of Unicode alphanumeric characters (letters plus digits
                plus underscore).
                With LOCALE, it will match the set [0-9_] plus characters
defined
                as letters for the current locale.
        \W      Matches the complement of \w.
        \\      Matches a literal backslash.

    This module exports the following functions:
        match     Match a regular expression pattern to the beginning of a
string.
        fullmatch Match a regular expression pattern to all of a string.
        search    Search a string for the presence of a pattern.
        sub       Substitute occurrences of a pattern found in a string.
        subn      Same as sub, but also return the number of substitutions made.
        split     Split a string by the occurrences of a pattern.
        findall   Find all occurrences of a pattern in a string.
        finditer  Return an iterator yielding a match object for each match.
        compile   Compile a pattern into a RegexObject.
        purge     Clear the regular expression cache.
        escape    Backslash all non-alphanumerics in a string.

    Some of the functions in this module takes flags as optional parameters:
        A  ASCII       For string patterns, make \w, \W, \b, \B, \d, \D
                       match the corresponding ASCII character categories
                       (rather than the whole Unicode categories, which is the
                       default).
                       For bytes patterns, this flag is the only available
                       behaviour and needn't be specified.
        I  IGNORECASE  Perform case-insensitive matching.
        L  LOCALE      Make \w, \W, \b, \B, dependent on the current locale.
        M  MULTILINE   "^" matches the beginning of lines (after a newline)
                       as well as the string.
                       "$" matches the end of lines (before a newline) as well
                       as the end of the string.
        S  DOTALL      "." matches any character at all, including the newline.
        X  VERBOSE     Ignore whitespace and comments for nicer looking RE's.
        U  UNICODE     For compatibility only. Ignored for string patterns (it
                       is the default), and forbidden for bytes patterns.
```

```
    This module also defines an exception 'error'.

CLASSES
    builtins.Exception(builtins.BaseException)
        sre_constants.error

    class error(builtins.Exception)
     |  Exception raised for invalid regular expressions.
     |
     |  Attributes:
     |
     |      msg: The unformatted error message
     |      pattern: The regular expression pattern
     |      pos: The index in the pattern where compilation failed (may be None)
     |      lineno: The line corresponding to pos (may be None)
     |      colno: The column corresponding to pos (may be None)
     |
     |  Method resolution order:
     |      error
     |      builtins.Exception
     |      builtins.BaseException
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __init__(self, msg, pattern=None, pos=None)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __weakref__
     |      list of weak references to the object (if defined)
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from builtins.Exception:
     |
     |  __new__(*args, **kwargs) from builtins.type
     |      Create and return a new object.  See help(type) for accurate
signature.
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from builtins.BaseException:
     |
     |  __delattr__(self, name, /)
     |      Implement delattr(self, name).
     |
     |  __getattribute__(self, name, /)
```

```
 |      Return getattr(self, name).
 |
 |  __reduce__(…)
 |      helper for pickle
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __setattr__(self, name, value, /)
 |      Implement setattr(self, name, value).
 |
 |  __setstate__(…)
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  with_traceback(…)
 |      Exception.with_traceback(tb) --
 |      set self.__traceback__ to tb and return self.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from builtins.BaseException:
 |
 |  __cause__
 |      exception cause
 |
 |  __context__
 |      exception context
 |
 |  __dict__
 |
 |  __suppress_context__
 |
 |  __traceback__
 |
 |  args

FUNCTIONS
    compile(pattern, flags=0)
        Compile a regular expression pattern, returning a pattern object.

    escape(pattern)
        Escape all the characters in pattern except ASCII letters, numbers and
'_'.

    findall(pattern, string, flags=0)
        Return a list of all non-overlapping matches in the string.
```

```
    If one or more capturing groups are present in the pattern, return
    a list of groups; this will be a list of tuples if the pattern
    has more than one group.

    Empty matches are included in the result.

finditer(pattern, string, flags=0)
    Return an iterator over all non-overlapping matches in the
    string.  For each match, the iterator returns a match object.

    Empty matches are included in the result.

fullmatch(pattern, string, flags=0)
    Try to apply the pattern to all of the string, returning
    a match object, or None if no match was found.

match(pattern, string, flags=0)
    Try to apply the pattern at the start of the string, returning
    a match object, or None if no match was found.

purge()
    Clear the regular expression caches

search(pattern, string, flags=0)
    Scan through string looking for a match to the pattern, returning
    a match object, or None if no match was found.

split(pattern, string, maxsplit=0, flags=0)
    Split the source string by the occurrences of the pattern,
    returning a list containing the resulting substrings.  If
    capturing parentheses are used in pattern, then the text of all
    groups in the pattern are also returned as part of the resulting
    list.  If maxsplit is nonzero, at most maxsplit splits occur,
    and the remainder of the string is returned as the final element
    of the list.

sub(pattern, repl, string, count=0, flags=0)
    Return the string obtained by replacing the leftmost
    non-overlapping occurrences of the pattern in string by the
    replacement repl.  repl can be either a string or a callable;
    if a string, backslash escapes in it are processed.  If it is
    a callable, it's passed the match object and must return
    a replacement string to be used.

subn(pattern, repl, string, count=0, flags=0)
    Return a 2-tuple containing (new_string, number).
    new_string is the string obtained by replacing the leftmost
    non-overlapping occurrences of the pattern in the source
```

```
        string by the replacement repl.  number is the number of
        substitutions that were made. repl can be either a string or a
        callable; if a string, backslash escapes in it are processed.
        If it is a callable, it's passed the match object and must
        return a replacement string to be used.

    template(pattern, flags=0)
        Compile a template pattern, returning a pattern object

DATA
    A = <RegexFlag.ASCII: 256>
    ASCII = <RegexFlag.ASCII: 256>
    DOTALL = <RegexFlag.DOTALL: 16>
    I = <RegexFlag.IGNORECASE: 2>
    IGNORECASE = <RegexFlag.IGNORECASE: 2>
    L = <RegexFlag.LOCALE: 4>
    LOCALE = <RegexFlag.LOCALE: 4>
    M = <RegexFlag.MULTILINE: 8>
    MULTILINE = <RegexFlag.MULTILINE: 8>
    S = <RegexFlag.DOTALL: 16>
    U = <RegexFlag.UNICODE: 32>
    UNICODE = <RegexFlag.UNICODE: 32>
    VERBOSE = <RegexFlag.VERBOSE: 64>
    X = <RegexFlag.VERBOSE: 64>
    __all__ = ['match', 'fullmatch', 'search', 'sub', 'subn', 'split', 'fi…

VERSION
    2.2.1

FILE
    /home/alice/anaconda3/lib/python3.6/re.py
```

2/ La première fonction que nous explorons est la fonction `findall`, quels sont les paramètres de cette fonction ? Quel est le type d'objet renvoyé ?

[ ]:
```
# paramètre 1 :
# paramètre 2 :
# paramètre 3 : (valeur par défaut : )
# valeur de retour :
```

3/ utiliser la fonction `findall` pour stocker dans les variables correspondantes :

- la liste des mots du texte

- la liste des mots du texte de trois lettres et moins
- la liste des nombres du texte
- la liste des adresses email présentes dans ce texte
- la liste des mots commençant par une majuscule
-

1.1 la liste des mots de trois lettres et moins

[4]:
```
# le texte dans lequel on cherche les motifs est le suivant :
text = "Si vous souhaitez procéder à votre inscription (date limite : le 24␣
 ↪septembre 2020), vous devez \n 1/ envoyer un mail au secrétariat :␣
 ↪secretariat@sorbonne.fr \n 2/ envoyer un mail au professeur principal :␣
 ↪professeur@sorbonne.fr (ou professeur@gmail.com) \n 3/ vous rendre à la␣
 ↪Sorbonne rue Serpente "

# affichez le texte :

# words contient les mots du texte (42 éléments)
words =
help(words)
print(words)

# numbers contient les nombres du texte (5 éléments)
#numbers = # à compléter
print(numbers)

# email contient les emails du texte (3 éléments)
#emails =  # à compléter
print(emails)

# capitalized contient les mots commençant par une majuscule (3 éléments)
#capitalized =  # à compléter
print(capitalized)

# small_words contient les mots de trois lettres et moins (16 éléments)
#small_words =  # à compléter
print(small_words)
```

```
Help on list object:

class list(object)
 |  list() -> new empty list
 |  list(iterable) -> new list initialized from iterable's items
 |
 |  Methods defined here:
```

```
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __delitem__(self, key, /)
|      Delete self[key].
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(…)
|      x.__getitem__(y) <==> x[y]
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __iadd__(self, value, /)
|      Implement self+=value.
|
|  __imul__(self, value, /)
|      Implement self*=value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
```

```
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __reversed__(…)
 |      L.__reversed__() -- return a reverse iterator over the list
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __setitem__(self, key, value, /)
 |      Set self[key] to value.
 |
 |  __sizeof__(…)
 |      L.__sizeof__() -- size of L in memory, in bytes
 |
 |  append(…)
 |      L.append(object) -> None -- append object to end
 |
 |  clear(…)
 |      L.clear() -> None -- remove all items from L
 |
 |  copy(…)
 |      L.copy() -> list -- a shallow copy of L
 |
 |  count(…)
 |      L.count(value) -> integer -- return number of occurrences of value
 |
 |  extend(…)
 |      L.extend(iterable) -> None -- extend list by appending elements from the
iterable
 |
 |  index(…)
 |      L.index(value, [start, [stop]]) -> integer -- return first index of
value.
 |      Raises ValueError if the value is not present.
 |
 |  insert(…)
 |      L.insert(index, object) -- insert object before index
 |
 |  pop(…)
 |      L.pop([index]) -> item -- remove and return item at index (default
```

```
last).
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(…)
 |      L.remove(value) -> None -- remove first occurrence of value.
 |      Raises ValueError if the value is not present.
 |
 |  reverse(…)
 |      L.reverse() -- reverse *IN PLACE*
 |
 |  sort(…)
 |      L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None

['Si', 'vous', 'souhaitez', 'procéder', 'à', 'votre', 'inscription', 'date',
'limite', 'le', '24', 'septembre', '2020', 'vous', 'devez', '1', 'envoyer',
'un', 'mail', 'au', 'secrétariat', 'secretariat', 'sorbonne', 'fr', '2',
'envoyer', 'un', 'mail', 'au', 'professeur', 'principal', 'professeur',
'sorbonne', 'fr', 'ou', 'professeur', 'gmail', 'com', '3', 'vous', 'rendre',
'à', 'la', 'Sorbonne', 'rue', 'Serpente']


↪----------------------------------------------------------------------------

       NameError                                 Traceback (most recent call last)

       <ipython-input-4-366b8367e3f2> in <module>
        11 # numbers contient les nombres du texte (5 éléments)
        12 #numbers = # à compléter
   ---> 13 print(numbers)
        14
        15 # email contient les emails du texte (3 éléments)


       NameError: name 'numbers' is not defined
```

4/ Les parenthèses permettent de former des groupes dans une expression régulière. Utilisées avec la fonction `findall`, elles permettent d'isoler les différents segments.
4.1/ Utilisez les parenthèses pour isoler la première (identifiant) et la seconde partie (nom de domaine) des adresses email et stockez le résultat dans `email_tuples`.

4.2/ Parcourez `email_tuples` et stockez dans `identifiants` la liste des identifiants et dans `noms_de_domaine` la liste des noms de domains présents dans ce texte.

[ ]:
```
email_tuples =  # à compléter
# résultat attendu : email_tuples = [('secretariat', 'sorbonne.fr'),␣
 ↪('professeur', 'sorbonne.fr'), ('professeur', 'gmail.com')]

identifiants = []
noms_de_domaine = []


# for i in range ...
# ...à compléter
#

print(identifiants)
# résultat attendu : identifiants = ['secretariat', 'professeur', 'professeur']


print(noms_de_domaine)
# résultat attendu : noms_de_domaine = ['sorbonne.fr', 'sorbonne.fr', 'gmail.
 ↪com']
```

5/ Utilisez l'expression `email_tuples` pour extraire les identifiants et noms de domaine du fichier emails_etudiants.txt

---

[ ]:
```
# ouverture du fichier
# f = ... à compléter

# définition de email_tuples
email_tuples =  # à compléter

identifiants = []
noms_de_domaine = []

# for i in range ...
# ...à compléter
#

print(identifiants)

print(noms_de_domaine)
```