Regular Expressions
Regexps

PROF. JACQUES SAVOY
UNIVERSITY OF NEUCHATEL

String Manipulation

Perform operations with the same string encoding
    UTF-8 with UTF-8, or ASCII vs. ASCII (or `str` and `bytes`).

Very useful to process textual data. Many applications.

Use the `re` package (`>>> import re`)

Usually, two strings: a pattern (string, regexp) to be searched into a (longer) second string (e.g., one line).

Require some special characters with the backslash (\) as in "`\n`" or '`\n`'

Using the raw string notation
  e.g., `r'\n'` means two characters (`\ n`)

Search with Regexp

Simplest pattern: a string such as `'film'`

Search for the first occurrence a regular expression in a line

`re.search(aPattern, aLine, flags)`

return `None` or a match object

Example:

```
>>> re.search('film', 'a beautiful film!')
   <re.Match object; span=(12, 16), match='film'>
```

The pattern `film` is inside the string (at least once)

`re.search()` differs from `re.match()`

Match with Regexp

The function `match()` searches from the *beginning* of the line (string)

`re.match(aPattern, aLine, flags)`

return `None` or a match object

Example:

```
>>> re.match('beau', 'beautiful film!')

  <_sre.SRE_Match object; span=(0,4), match='beau'>
```

The pattern `film` is at the beginning of the line.

```
>>> re.match('film', 'beautiful film!')
```

>>> No answer.

Search with Regexp

More efficient solution (when you need to repeat it) vs. using module-level function

```
re.compile(aPattern, flags)     # compiled regular expressions
```

use as:

```
aProg   = re.compile(aPattern)
aResult = aProg.search(aLine)
```

Example:

```
>>> myProg = re.compile('film')
>>> myProg.search('a good film!')
   <_sre.SRE_Match object; span=(7,11), match='film'>
>>> myProg.search('a good movie!')
>>> # none
```

Flags

Ignore the difference uppercase / lowercase

Add the flags `re.I`

```
>>> myProg = re.compile('film', re.I)
>>> myProg.search('a good FILM!')
   <_sre.SRE_Match object; span=(7,11), match='FILM'>
```

Case sensitive

```
>>> myProg = re.compile('film')
>>> myProg.search('a good FILM!')
>>>
```

Substitute

Replace the occurrence(s) of a string by another one.

```
re.sub(aPattern, aRepl, aLine, count, flags)
```

return a string (with or without replacement(s))

```
>>> re.sub('film', 'movie', 'a good film!')
    'a good movie!'
```

Case sensitive

```
>>> re.sub('film', 'movie', 'a good film! FILM, and film')
    'a good movie! FILM, and movie'
>>> re.sub('film', 'movie', 'a good film! FILM, and film',
0, re.I)
    'a good movie! movie, and movie'
```

Text Normalization

Expand contractions in English

Negations

```
>>> re.sub("n't", " not", "I didn't do this. Ann hasn't it.")
   'I did not do this. Ann has not it.'
```

Future tense

```
>>> re.sub("'ll", " will", "I'll do it, and we'll sing")
   'I will do it, and we will sing'
```

Other forms: "we've", "Paul's book", "11th", ...

Find All

Extract all occurrences of a given pattern.

`re.findall(aPattern, aLine, flags)`

returns a list of strings respecting the pattern

```
>>> re.findall('film', 'a beautiful film! FILM, and Film')
    ['film']  # a list
>>> re.findall('film', 'a good film! FILM, and Film', re.I)
    ['film', 'FILM', 'Film']
```

Find All

And with the diacritics…

```
>>> aLine = 'que de belles comédies et comedies'

>>> re.findall("[\w]+", aLine)
    ['que', 'de', 'belles', 'comédies', 'et', 'comedies']
```

But with some limitations…

```
>>> re.findall("[A-Za-z]+", aLine)
['que', 'de', 'belles', 'com', 'dies', 'et', 'comedies']
```

Result

A simple function to display the result of a search.

```
def displayMatch(aMatch):
  if aMatch is None:
     return None
  return '<Match: %r>' % (aMatch.group())
```

Example:

```
>>> myProg = re.compile('film')
>>> displayMatch(myProg.search('this Film is good film!'))
   "<Match: 'film'>"
```

Result

More about the result

```
>>> myProg = re.compile('film')
>>> aRes = myProg.search('this is a good film!')
```

The match, the starting, and the ending position

```
>>> aRes.group(0)
   'film'
>>> aRes.start(0)
   15
>>> aRes.end(0)
   19
```

More Complex Regexps

Specify a list of possible characters inside `[]`

Example: *one* of the digits `[01234556789]`
or in short `[0-9]`

Example:

```
>>> myProg = re.compile('[Ff]ilm')
>>> displayMatch(myProg.search('this Film is good film!'))
   "<Match: 'Film'>"
```

More Complex Regexps

Regexp expressions can contain both ordinary and special characters.

Previously, we have the two signs `[]` and – (e.g. `[0-9]`)

The ^ is a special character meaning not the following characters.

Example: Not a vowel `[^aeiou]`

For any character, write `.` (except newline) (another special character)

Example:

```
>>> myProg = re.compile('[^0123456789]')
>>> displayMatch(myProg.search('2001Troy!'))
   "<Match: 'T'>"
```

More Complex Regexps

Repetition of the previous regexp

- * means 0 or more repetitions
- ? means 0 or one repetition
- + means 1 or more repetitions

To avoid the interpretation of a meta-character (' ? ^ etc.), add \ before it.

Example

```
>>> aPat = '[$]? ?[0-9]+\.[0-9]*'
>>> re.findall(aPat, '$2.50, $3 1.345 or .95')
   ['$2.50', ' 1.345']
>>> re.findall(aPat, '$2.50, $3.0 1.345 or 0.95')
   ['$2.50', '$3.0', '1.345', '0.95']
```

More Complex Regexps

Detect and change the URLs with a predefined string.

```
>>> aTweet = "Weak pathetic Democrat Mayor!! https://t.co/dehIDMwgul"
>>> aMatch = re.search('http[s]?://[A-Za-z0-9/\.]* ?', aTweet, re.I)

>>> if (aMatch):
      aPos = max(aMatch.start()-1, 0)
      aTweet = aTweet[:aPos] + " urllink " + aTweet[aMatch.end():]
>>> aTweet   # I only remove the first URL
   'Weak pathetic Democrat Mayor!! urllink '
```

More Complex Regexps

But *greedy* evaluation! As much as possible.

Example

```
>>> aPat='<[A-Z][A-Za-z]*>.*</[A-Z][A-Za-z]*>'
>>> re.findall(aPat,'<Top>Title</Top> <Head>Headline</Head>')
   ['<Top>Title</Top> <Head>Headline</Head>']
```

A single occurrence!

```
>>> aPat='<[A-Z]>.*</[A-Z]>'    # simpler pattern
>>> re.findall(aPat,'<T>Title</T> Tintin <H>Head</H>')
   ['<T>Title</T> Tintin <H>Head</H>']
```

More Complex Regexps

To avoid the *greedy* evaluation, add `?`

```
>>> aPat='<[A-Z]>.*?</[A-Z]>'
>>> re.findall(aPat,'<T>Title</T> Tintin <H>Head</H>')
  ['<T>Title</T>', '<H>Head</H>']
```

And if you need to extract the content specified by the tag
(e.g., the string 'Title' or 'Head')?

Need to define group(s) with the `()`

More Complex Regexps

To extract the element inside the tags

```
>>> aPat = '<[A-Za-z]>(.*?)</[A-Za-z]>'
>>> re.findall(aPat,'<T>Title</T>  <H>Head</H>')
['Title', 'Head']
>>> re.findall(aPat,'<T>Title</t> tintin <h>Head</h>')
['Title', 'Head']
```

More Complex Regexps

- More than one group…
- Decompose a price into an integer part and the decimal one.
- The first group for the integer part and the second one for the fractional.

Example

```
>>> aPat = '[$]?([0-9]+)\.([0-9]*)'
>>> re.findall(aPat, '$2.50, $3 1.345 or .95')
   [('2', '50'), ('1', '345')]
```

More Complex Regexps

More meta characters

- Indicate a digit by `\d` (equivalent to `[0-9]`)
- Not a digit `\D`
- Indicate a letter (word character) by `\w`
- Space by `\s` (or `\t\n\r\f\v`) (and negation with \S)
- Word boundary with `\\b` (or `[^A-Za-z0-9_]`)

Example of a tokenizer

```
>>> re.findall('[\w]+', 'Jéan I le bon.')
   ['Jéan', 'I', 'le', 'bon']
```

Simple Tokenizer

Tokenizer: split a text into tokens (words)

```
>>> aLine = "A computer!!! IBM-360 IBM.360 IBM_360 IBM360."
>>> re.findall('[\w]+', aLine)
   ['A', 'computer', 'IBM', '360', 'IBM', '360', 'IBM_360', 'IBM360']
>>> re.findall('[\w-]+', aLine)
   ['A', 'computer', 'IBM-360', 'IBM', '360', 'IBM_360', 'IBM360']
>>> re.findall('[\w\.-]+', aLine)
   ['A', 'computer', 'IBM-360', 'IBM.360', 'IBM_360', 'IBM360.']
```

Regexp Tokenizer

More on tokenizer

```
>>> aLine = "Peter's book? THE price is $32.90."
>>> re.findall("\w+", aLine)
   ['Peter', 's', 'book', 'THE', 'price', 'is', '32', '90']
>>> re.findall("\w+['\w+]*", aLine)
   ["Peter's", 'book', 'THE', 'price', 'is', '32', '90']
```

Extract the non-alphanumerical sequences

```
>>> re.findall("[^\w\s]", aLine)     # not A-Za-z0-9_ and spaces
   ["'", '?', '$', '.', '.']
>>> re.findall("[^\w\s]", 'Peter. !!!')
   ['.', '!', '!', '!']
>>> re.findall("[^\w\s]+", 'Peter. !!!')  # sequence length > 0
   ['.', '!!!']
```

Regexp Tokenizer

More on tokenizer: Use the | for the OR operator

```
>>> aLine = "this is Peter's book at $32.90 and Ann's pen."
>>> re.findall("\w+|\$[\d\.]+", aLine)
   ['this', 'is', 'Peter', 's', 'book', 'at', '$32.90', 'and', 'Ann', 's',
'pen']
```

Not fully clear with `I've, Peter's, didn't` …

```
>>> aLine = "I've Peter's book that didn't cost $32.90, yes."
>>> re.findall("[\w']+|\$[\d\.]+", aLine)
   ["I've", "Peter's", 'book', 'that', "didn't", 'cost', '$32.90', 'yes']
```

Regexp Tokenizer

More on tokenizer

```
>>> aLine = "this is Peter's book at $32.90 and Ann's pen."
>>> re.findall(r"\w+(?:'\w+)?|[^\w\s]", aLine)
 ['this', 'is', "Peter's", 'book', 'and', "Ann's", 'pen', '.']
>>>  re.findall(r"\w+|\$[\d\.]+|\S+", aLine)
 ['this', 'is', 'Peter', "'s", 'book', 'at', '$32.90', 'and', 'Ann', "'s",
 'pen', '.']
>>> aLine = "A computer!!! IBM-360 IBM.360 IBM_360 IBM360."
>>> re.findall(r"\w+|\$[\d\.]+|\S+", aLine)
['A', 'computer', '!!!', 'IBM', '-360', 'IBM', '.360', 'IBM_360', 'IBM360',
'.']
```

More Complex Regexps

Specify the number of previous occurrences.

- Use the `{m, n}` notation
- From a minimum `m` characters, and up to `n`.

Example

- `\w{1,3}` a sequence between 1 and 3 letters
- One parameter could be missing (e.g., `{,3}`)

Adverbs

Extract all words ending with –ly

But the length must be larger than 3 (before –ly)

```
>>> aLine = "this costly, greatly and poly book"

>>> aPattern = re.compile(u' (\w{3,})+ly[ ,\.]')

>>> aPattern.findall(aLine)
    ['cost', 'great']
```

More Information

To test your regexps:

https://regex101.com/

The Python 3 documentation on regexp

https://docs.python.org/3/library/re.html

More Complex Formulations

A few problems with the slash and backslash
the r'...' is denoted raw string and Python will not interpret the string

```
>>> aString = '<a> didn''t eat/drink this!</a>'
>>> aMatch = re.search('/', aString)       # match
>>> aMatch = re.search('\/', aString)      # match
>>> aMatch = re.search(r'/', aString)      # match (raw string)

>>> aString = '<a> didn''t eat\drink this!<\a>'
>>> aMatch = re.search('\', aString)       # impossible
>>> aMatch = re.search(r'\\', aString)     # match (raw string)
>>> aMatch = re.search('\\', aString)      # impossible
>>> aMatch = re.search('\\\\', aString)    # match
```