Pattern matching with regular expressions

Andreas Busschop & Michaël Vanhelmont

Finding Patterns of Text without Regular Expressions

- find telephone nr.
- vb: 415-555-4242
- 12 long
- starts with 3 digits
- a ‘-’
- …

```
def isPhoneNumber(text):
        if len(text) != 12:
        return False
    for i in range(0, 3):
                if not text[i].isdecimal():
            return False
        if text[3] != '-':
        return False
    for i in range(4, 7):
                if not text[i].isdecimal():
            return False
        if text[7] != '-':
        return False
    for i in range(8, 12):
                if not text[i].isdecimal():
            return False
        return True
```

K.H.Kempen en Lessius bundelen de krachten en worden *more*.

Finding Patterns of Text without Regular Expressions

```
def isPhoneNumber(text):
        if len(text) != 12:
        return False
    for i in range(0, 3):
                if not
text[i].isdecimal():
            return False
        if text[3] != '-':
        return False
    for i in range(4, 7):
                if not
text[i].isdecimal():
            return False
        if text[7] != '-':
        return False
    for i in range(8, 12):
                if not
text[i].isdecimal():
            return False
        return True
```

```
print('415-555-4242 is a phone number:')
print(isPhoneNumber('415-555-4242'))
print('Moshi moshi is a phone number:')
print(isPhoneNumber('Moshi moshi'))
```

```
415-555-4242 is a phone number:
True
Moshi moshi is a phone number:
False
```

Finding Patterns of Text without Regular Expressions

```
message = 'Call me at 415-555-1011 tomorrow. 415-555-9999 is my office.'
for i in range(len(message)):
        chunk = message[i:i+12]
        if isPhoneNumber(chunk):
                print('Phone number found: ' + chunk)
print('Done')
```

```
Phone number found: 415-555-1011
Phone number found: 415-555-9999
Done
```

Finding Patterns of Text with Regular Expressions

- Without Regular Expressions:
  - isPhoneNumber() has 17 lines
  - only 1 pattern: xxx-xxx-xxxx (415-555-1011)
  - no 415.555.4242 or (415) 555-4242 …
- Solution → Regular Expressions:

  Regular expressions, called regexes for short, are descriptions for a pattern of text

Finding Patterns of Text with Regular Expressions

```
>>> import re
```

- **import module RegularExpressions**

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
```

- **“re.compile() ” geeft een Regex object**

```
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
```

- **“.search” zoekt in de string naar de regex**

```
>>> print('Phone number found: ' + mo.group())
```

- **“mo.group()” geeft de gevonden regex weer (mo is afkorting voor Match Object)| geen match: mo = None**

Finding Patterns of Text with Regular Expressions

“Phone number found: 415-555-4242”

But what if you print a match object?

```
>>> print(mo)
<_sre.SRE_Match object; span=(13, 25), match='415-555-4242'>
>>> print(mo) ← no match
None
```

```
.compile(r'\\')
.search('Robo-Cop e\\ats baby food.')
<_sre.SRE_Match object; span=(10, 11), match='\\'>
```

Grouping with Parentheses

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
```

.group()

```
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242'
```

.groups()

```
>>> mo.groups()
('415', '555-4242')
>>> areaCode, mainNumber = mo.groups()
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

Grouping with Parentheses

```
>>> phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My phone number is (415) 555-4242.')
>>> mo.group(1)
'(415)'
>>> mo.group(2)
'555-4242'
```

Matching Multiple Groups with the Pipe

pipe

```
>>> heroRegex = re.compile (r'Batman|Tina Fey')
>>> mo1 = heroRegex.search('Batman and Tina Fey.')
>>> mo1.group()
'Batman'
>>> mo2 = heroRegex.search('Tina Fey and Batman.')
>>> mo2.group()
'Tina Fey'
```

..(pipe)

```
>>> batRegex = re.compile(r'Bat(man|mobile|copter|bat)')
>>> mo = batRegex.search('Batmobile lost a wheel')
>>> mo.group()
'Batmobile'
>>> mo.group(1)
'mobile'
```

Optional Matching with the Question Mark

match optional

```
>>> batRegex = re.compile(r'Bat(wo)?man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'
>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
```

Matching Zero or More with the Star

match 0, 1 or more times

```
>>> batRegex = re.compile(r'Bat(wo)*man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'
>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
>>> mo3 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'
```

Matching One or More with the Plus

match 1 or more times

```
>>> batRegex = re.compile(r'Bat(wo)+man')
>>> mo1 = batRegex.search('The Adventures of Batwoman')
>>> mo1.group()
'Batwoman'
>>> mo2 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo2.group()
'Batwowowowoman'
156 Chapter 7
>>> mo3 = batRegex.search('The Adventures of Batman')
>>> mo3 == None
True
```

Matching Specific Repetitions with Curly Brackets

(Ha){3} == (Ha)(Ha)(Ha)

(Ha){min,max}

(Ha){3,} == ((Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha))|...

(Ha){,5} == ((Ha))|((Ha)(Ha))|...|((Ha)(Ha)(Ha)(Ha)(Ha))

1 2

5

(Ha){3,5} == ((Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha)(Ha))

3 4

5

K.H.Kempen en Lessius bundelen de krachten en worden *more*.

Greedy and Nongreedy Matching

greedy (default)

```
>>> greedyHaRegex =
   re.compile(r'(Ha){3,5}')
>>> mo1 =
   greedyHaRegex.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'
```

non-greedy

```
>>> nongreedyHaRegex =
   re.compile(r'(Ha){3,5}?')
>>> mo2 =
   nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

The findall() Method

What if we want to find ALL matches? → findall()

More than 1 match with .search:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('Cell: 415-555-9999 Work: 212-555-0000')
>>> mo.group()
'415-555-9999'
```

The findall() Method

No groups

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
['415-555-9999', '212-555-0000']
```

→ **returns a list of matches**

Groups

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)')
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
[('415', '555', '1122'), ('212', '555', '0000')]
```

→ **returns list of tuples**

Character Classes

\d = numeric digit or short for (0|1|2|3|4|5|6|7|8|9)
\D = NOT \d
\w = letter, numeric digit, underscore (“word” characters)
\W = NOT \w
\s = space, tab (\t), newline (\n) (“space” characters)
\S = NOT \s

Character classes

```
>>> xmasRegex = re.compile(r'\d+\s\w+')
>>> xmasRegex.findall('12 drummers, 11 pipers, 10 lords, 9 ladies, 8 maids,
  7 swans, 6 geese, 5 rings, 4 birds, 3 hens, 2 doves, 1 partridge')

['12 drummers', '11 pipers', '10 lords', '9 ladies', '8 maids', '7 swans', '6
geese', '5 rings', '4 birds', '3 hens', '2 doves', '1 partridge']
```

Making Your Own Character Classes

With “[ ]”, you can make your own class.

All vowels:
[aeiouAEIOU]

```
>>> vowelRegex = re.compile(r'[aeiouAEIOU]')
>>> vowelRegex.findall('RoboCop eats baby food. BABY FOOD.')
['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o', 'A', 'O', 'O']
```

Making Your Own Character Classes

Ranges:
[0-5]
[a-zA-Z0-9]

No need to escape . * ? () ...
[0-5.]

But “-”:
[0-5\-]

negative character class:
[^aeiouAEIOU]

The Caret and Dollar Sign Characters (^ and $)

^ : string has to begin with pattern

```
>>> beginsWithHello = re.compile(r'^Hello')
>>> beginsWithHello.search('Hello world!')
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>> beginsWithHello.search('He said Hello') == None
True
```

The Caret and Dollar Sign Characters (^ and $)

$: string has to end with pattern

```
>>> endsWithNumber = re.compile(r'\d$')
>>> endsWithNumber.search('Your number is 42')
<_sre.SRE_Match object; span=(16, 17), match='2'>
>>> endsWithNumber.search('42 is your number') == None
True
```

The Wildcard Character

“.” in regex = wildcard = any character but newline (\n)

```
>>> atRegex = re.compile(r'.at')
>>> atRegex.findall('The cat in the hat sat on the flat mat.')
['cat', 'hat', 'sat', 'lat', 'mat']
```

- Escape \. if you want to find a dot character

Matching Everything with Dot-Star

.* = any character but newline + zero or more of the preceding character

```
>>> nameRegex = re.compile(r'First Name: (.*) Last Name: (.*)')
>>> mo = nameRegex.search('First Name: Al Last Name: Sweigart')
>>> mo.group(1)
'Al'
>>> mo.group(2)
'Sweigart'
```

Matching Everything with Dot-Star

Non greedy:

```
>>> nongreedyRegex = re.compile(r'<.*?>')
>>> mo = nongreedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man>'
```

Greedy:

```
>>> greedyRegex = re.compile(r'<.*>')
>>> mo = greedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man> for dinner.>'
```

Matching Newlines with the Dot Character

No newline:

```
>>> noNewlineRegex = re.compile('.*')
>>> noNewlineRegex.search('Serve the public trust.\nProtect the innocent.
\nUphold the law.').group()
'Serve the public trust.'
```

Newline:

```
>>> newlineRegex = re.compile('.*', re.DOTALL)
>>> newlineRegex.search('Serve the public trust.\nProtect the innocent.
\nUphold the law.').group()
'Serve the public trust.\nProtect the innocent.\nUphold the law.'
```

Review of Regex Symbols

\d ⇒ digit	\w ⇒ word characters (1,w,_)	\s ⇒ space,\t,\n	\D, \W, \S ⇒ NOT \d,\w,\s
| ⇒ or	(x)? ⇒ 0 or 1 (optional)	(x)* ⇒ 0 or more	(x)+ ⇒ 1 or more
x{exactly}	x{min,max}	{x,y}? or *? ⇒ nongreedy	
^x ⇒ begin	x$ ⇒ end		
. ⇒ any (exept \n)		[abc] ⇒ all characters in class	[^abc] ⇒ all characters but class

Case-Insensitive Matching

re.I or re.IGNORECASE as second argument of .compile:

```
>>> robocop = re.compile(r'robocop', re.I)
>>> robocop.search('RoboCop is part man, part machine, all cop.').group()
'RoboCop'
>>> robocop.search('ROBOCOP protects the innocent.').group()
'ROBOCOP'
>>> robocop.search('Al, why does your programming book talk about
  robocop so much?').group()
'robocop'
```

Substituting String with the sub() Method

```
>>> namesRegex = re.compile(r'Agent \w+')
>>> namesRegex.sub('CENSORED', 'Agent Alice gave the secret documents to Agent Bob.')
'CENSORED gave the secret documents to CENSORED.'

>>> agentNamesRegex = re.compile(r'Agent (\w)\w*')
>>> agentNamesRegex.sub(r'\1****', 'Agent Alice told Agent Carol that Agent Eve knew Agent Bob was a double agent.')
A**** told C**** that E**** knew B**** was a double agent.'
```

- **\1, \2, \3… = “Enter the text of group 1, 2, 3... in the substitution.”**

Managing Complex Regexes

```
phoneRegex = re.compile(r'''(
(\d{3}|\(\d{3}\))?            # area code

(\s|-|\.)?                    # separator
\d{3}                         # first 3 digits
(\s|-|\.)                     # separator
\d{4}                         # last 4 digits
(\s*(ext|x|ext.)\s*\d{2,5})?  # extension
)''', re.VERBOSE)
```

Combining re.IGNORECASE, re.DOTALL, and re.VERBOSE

Piping!

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL |
  re.VERBOSE)
```

Vragenronde

- What is the function that creates Regex objects?
- What does the search() method return?
- () en . hebben specifieke betekenissen in regex syntax. Hoe specifieer je een regex om een match te hebben met () en . karakters?
- What does the | character signify in regular expressions?
- What is the difference between {3} and {3,5} in regular expressions?
- How do you make a regular expression case-insensitive?

And now exercises…

trac.pbei.be: https://trac.pbei.be/wiki/RegularExpressions