Regular Expressions in Python

Max Floettmann

2013-15-01

Search and Replace

- Regular expressions are used to find specific patterns in texts

Regexs are a Language

- Very compact
- lots of thought per character
- implemented in many languages

Examples

Email adresses:

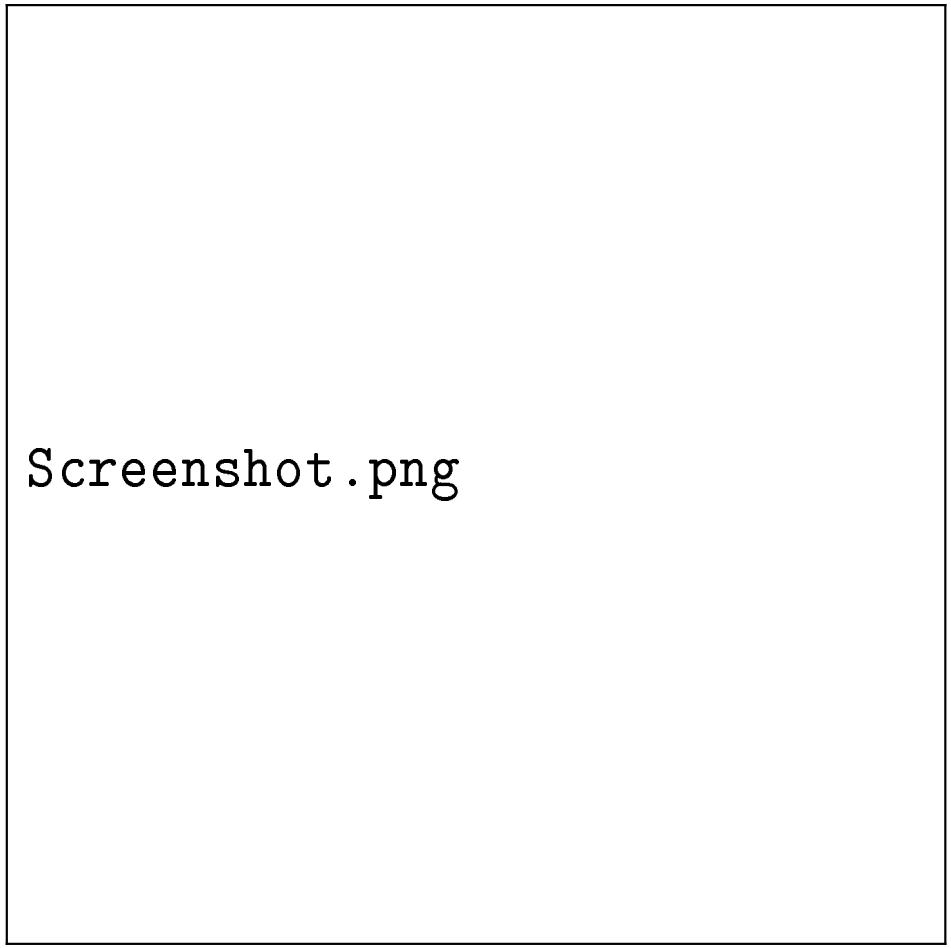
```
[a-z0-9!#$%&'*+/=?^_`{|}~-]+
(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)
*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+
[a-z0-9](?:[a-z0-9-]*[a-z0-9])?
```

HTML Tags:

```
<([A-Z][A-Z0-9]*)\b[^>]*>(.*?)</\1>
```

Adresscrawler

Scan websites for email adresses to send your job application to.



Matching Simple Text

- searching a special email adress in a document
- very limited
- use of special characters in regular expression enhances flexibility

Example (Simple Searching)

```
text = 'blablablubb@gmail.com'
Find('blablubb@gmail.com',text)
```

Introducing: The '.'

- a wildcard matching every possible character
- should be used seldomly because it is unspecific

Example (using the '.')

```
Find('blablablubb@gmail.com',text)
Find('..@..',text)
```

Character Classes

- What if we know we are looking for a character and not a number?
  - Define a set of possible characters.

Definition (notation)

[<characters>]

Example (defining classes)

```
result = re.findall('ilse[ae]igner@gmail.com',text)
email = re.findall(r'[a-z-.]+@[a-z-.]+',text)
```

Shorthand Character Classes

- Predefined sets of characters

Example (Most Important Classes)

```
space:   \s = [ \t\n\r\f\v]
word:    \w = [a-zA-Z0-9_]
decimal: \d = [0-9]
```

Important!

Case sensitive! Capital versions of the shorthands are the inverse set of characters. So "\S" matches every non whitespace character.

Repetitions

- Email adresses do not have fixed size
- repetition symbols enable us to match any number of characters

Different Symbols

- '*' 0 or more
- '+' 1 or more
- '{x[,y]}' between x and y

#'.*@.*' ? better not

Example (Repeating Characters)

```
text = 'blaaaablablubb@gmail.com'
Find('bla*',text)
Find('bla{1,3}',text)
re.findall('b+',text)
```

Alternatives

- some people want to protect themselves from getting spammed by using '[at]' instead of '@'
- we want to match both alternatives, so we use a logical or '|'

Example (Alternative Expressions:)

```
email = re.findall(r'([a-z-.]+)(@|\[at\])([a-z-.]+)',text)
```

Groups

- parentheses can be used to define groups in the expression
- groups can have two effects:
  1. define a term like in an algebraic expression
  2. store the result of the expression inside so that it can be used later (we will come to that in a minute)
- to suppress the second effect we can use the (?:<expression>) notation

Example (Groups and non Grouping Groups)

```
email = re.findall(r'([a-z-.]+)(@|(\[at\]))
                    ([a-z-.]+)',text)
email = re.findall(r'(?:[a-z-.]+)(?:@|(?:\[at\]))
                    (?:[a-z-.]+)',text)
```

Optional Text

- in some cases there are also spaces between the @ and the rest of the adress
- Solution: Optional text

Example (With Optional Spaces:)

```
email = re.findall(r'([a-z-.]+)\s?(@|(\[at\]))\s?
                    ([a-z-.]+)',text)
```

Flags

- sometimes people used an [AT] instead of [at]
- we can deal with this if we use the IGNORECASE flag

```
(?i)(<expression>)
```

Example (IGNORECASE)

```
email = re.findall(r'(?i)([a-z-.]+)\s?(@|\[at\])\s?
                     ([a-z-.]+)',text)',text)
```

Backreferences

- as mentioned before, the result of defined groups is stored for later use
- not only in the returned value, but also during evaluation of the string
- match only adresses that are similar to johndoe@johndoe.com

- Backreferences

```
test = 'johndoe@johndoe.com'
email = re.findall(r'(?i)([a-z-.]+\s?)(@|\[at\])
                    (\s?\1\.[a-z-.]+)',test)
```

Named Groups

- groups can get a name for easier backreferencing
- you can insert new groups without changing the rest of the expression
- Named Groups

```
email = re.findall(r'(?i)(?<=mailto:)(?P<name>:([a-z-.]+\s?
                                    (@|\[at\])(\s?(?P=name)\.[a-z-.]+)
```

Look Around You

- make sure some pattern is (not) there, but do not match it
- positive and negative look ahead and look behind

look ahead

- (?=)
- (?!)

look behind

- (?<=)
- (?<!)

Example (only use mailto links)

```
email = re.findall(r'(?i)(?<=mailto:)(?:([a-z-.]+\s?)(@|\[at\])
                        (\s?[a-z-.]+))',text)
```

Greedy vs. Non-Greedy

- look at the file again
- now we want to match HTML-tags
- greedy versions of repetition not very handy

Example (Match HTML Tags)

```
tags = re.findall(r'<+*>',text)
#better
tags = re.findall(r'<+*?>',text)
```

Python Implementation

- Module Import and Methods
  - regular expressions are implemented in the module “re”
  - after importing we have a number of methods to evaluate regular expressions at hand

Example (importing and using re module:)

```
import re
re.search('blablubb','blablablubb@gmail.com')
result = re.search('blablubb','blablablubb@gmail.com')
result.group()
```

- Methods
  - re.search
  - re.findall
  - re.compile
  - re.sub

Verbose / Free Spacing Mode

- A nicer way of writing complex regular expressions
- enables use of different lines and comments

Example (verbose mode)

```
pattern = r'
(?ix)                 #IGNORECASE and VERBOSE flags
(?<=mailto:)          #lookbehind to only parse mailto links
(?:([a-z-.]+)         #username
(\s?(?:@|\[at\]))     #the '@' or alternative against spammers
(\s?[a-z-.]+))        #hostname
'
```

Convenience Function

Example (this function prints the results of re.search(x,y):)

```
def Find(x,y):
  res = re.search(x,y)
  if res:
    print res.group()
  else:
    print 'not found'
```

Website

For our approach of spamming people we first have to pull the sourcecode of our group website:

Example (opening a URL:)

```
import urllib2
website = urllib2.urlopen('http://www2.hu-berlin.de/
                           biologie/theorybp
                           /index.php?goto=people')
text = website.read()
```