チュートリアル
補足スライド
３章

正規表現とは

* 正規表現とは？
  ▶文字列のパターンを表現する表記法

* 例えばどのようなもの？
  「^ABC」：行頭がABCで始まるもの
  URLとのマッチング：（URIの妥協した正規表現）
  「(([^:/?#]+):)?(//([^/?#]*))?([^?#]*)(¥?([^#]*))?(#(.*))?」

http: **//www.ics.uci.edu** **/pub/ietf/uri/**

* とても奥が深い。
  ▶それほど深入りせず、使い方に慣れるように簡単な例だけ。

正規表現の機能

* マッチング：
  対象の文字列が与えられたパターンにマッチするか調べる(T/F)

* キャプチャ：
  パターン中のあるパターンにマッチした文字列（サブマッチ）を後で参照できるように保存する。

pythonでの正規表現の使い方（基本）

```
#coding:utf-8
import re
#reモジュールをインポート

match_obj = re.match("Category","Category:イギリス")

if match_obj:
        print "aruyo"

match_obj = re.match("(.*?):(.*)", "Category:イギリス")

if match_obj:
        print "zenbu: ", match_obj.group(0)#
        print "first submatch: ", match_obj.group(1)
        print "second submatch: ", match_obj.group(2)
```

マッチング

キャプチャ

（補足）if match_obj: はどう解釈されるのか

* pythonのifは条件式がTrueかFalse(bool型)の値かで条件分岐する.
* **では、if match_objとはいったいどう解釈されるのか？**
* match_objには正規表現にマッチした場合、
  <_sre.SRE_Match object at 0x*******>（マッチオブジェクト）が返され、違う場合Noneが返される。
* pythonの「if x:」は条件式xをbool(x)で評価する。
  xが何のオブジェクトならFalseになるかは次の仕様となっている。
  None, False, 数値型におけるゼロ。例えば 0, 0L, 0.0, 0j 。
  空のシーケンス型。例えば '', (), [] 。 , 空のマッピング型。例えば {}
* したがって、bool(match_obj)はマッチしていればTrueで、マッチしていなければFalseと解釈される。

正規表現の基本機能

* 連接：
  「abcd」(abcdと連なる文字列)

* 選択：
  「a|c」（aまたはc)

* 繰り返し：
  「ba*」(bのあとaを0回以上繰り返す)

正規表現の特殊文字

メタキャラクタ	意味
.	任意の1文字
\w	英単語を構成する文字(a~z,A~Z,_,1~9)
\W	英単語を構成する文字以外
\s	空白文字(半角スペース,タブ,改行,キャリッジリターン)
\S	空白文字以外
\d	半角数字(0~9)
\D	半角数字以外
\b	単語の境界に一致
[xyz]	指定された文字のどれかに一致(この場合xyzのいずれかに一致)
[a-z]	マッチする文字の範囲を指定する表現(この場合aからzまで他には[1-9][A-Z]など文字コードが連続していれば使える。)
(pattern1|pattern2)	指定されたパターンのどれかにマッチする表現

* http://qiita.com/tossh/items/635aea9a529b9deb3038

練習問題1(マッチ)

* Q1. リスト：[‘aho’, ‘DOJI’, ‘MANUKE’]
  からアルファベットの大文字で書かれた文字列のみを抽出せよ.

* Q2. リスト：targets = [u"構文", u"解析", u"たのしい"]
  から、ひらがな（あ〜ん）で書かれた文字列のみを抽出せよ.

練習問題1のコード

```
#coding:utf-8
import re

targets1 = ['aho', 'DOJI', 'MANUKE']
targets2 = [u"構文", u"解析", u"たのしい"]

extracted = []
pattern1 = [redacted]
pattern2 = [redacted]
#pattern3 = u”[一-龠].*" # 漢字のみ（非厳密）を抽出

for target in targets:
        match_obj = re.match(pattern1,target1)
        if match_obj:
                extracted.append(target)

for line in extracted:
        print line
```

練習問題2(キャプチャ)

* 鍵括弧で囲まれた文字列を取得せよ.
* Q3. 作者はさっき、「下人が雨やみを待っていた」と書いた。

* Q4. しかしこの「すれば1」は、いつまでたっても、結局「すれば2」であった。
  ※最初のすれば1を抽出すること

練習問題2 コード

```
#coding:utf-8

import re

target1 = u"作者はさっき、「下人が雨やみを待っていた」
と書いた。"
target2 = u"しかしこの「すれば1」は、いつまでたって
も、結局「すれば2」であった。"

pattern = ur"[redacted]"

match_obj = re.match(pattern,target1)
if match_obj:
        print match_obj.group(1)
```

正しく動く正規表現か

* アイデア：
  1. 何かの文字列が続いて、
  2. その後鍵括弧「が来て、
  3. 次に何かの文字列（キャプチャ対象）がきて、
  4. 次に鍵括弧」が来て、
  5. 最後は何かの文字列が続く

* .*「(.*)」.*はどうだろうか.

* Q3はうまくいくがQ4は２個目の鍵括弧の中の文字列が入る.

* なぜか？

欲張り(greedy)

* .*「.*」.*の探索

しかしこの「すれば1」は、いつまでたっても、結局「すれば2」であった。

怠け者(lazy)

* .*?「.*?」.*の探索

しかしこの「すれば1」は、いつまでたっても、結局「すれば2」であった。

* lazy（.*?）を使うことで、最初のマッチで止まる。
  また、文字列を余計に探索しないためパフォーマンスがあがることもある.

* lazy版をQ4で試してください。
  また、.*?「.*」.*はどうなるか想像し、想定と同じ文字列を取得できるか試してください。

matchとsearchの違い

* match関数:
  先頭でマッチ

search関数:
マッチした位置を返す。

用途に応じて選択

```
import re

s = "ashi ga itai"

target = "ga"

match_obj = re.match(target, s)
if match_obj:
        print "aruyo"
else:
        print "naiyo"
match_obj = re.search(target, s)

if match_obj:
        print  match_obj.start()
```

参考文献等

* 正規表現技術入門（技術評論社）

http://qiita.com/keisuke-nakata/items/e0598b2c13807f102469

http://docs.python.jp/2/library/stdtypes.html#truth