正規表現による文字列処理

名古屋大学 情報基盤センター
情報基盤ネットワーク研究部門
基盤ネットワーク研究グループ
嶋田 創

ポイント

- 正規表現で文字列を自在に切り出せるようになる
- 以下あたりの文字列を切り出せるぐらいにはなって欲しい
  - 例1: n桁の数字と記号の特定の並び
  - 例2: 空白で区切られた英語3ワードのみからなる行
- 文字列操作と合わせてテキストファイルを処理できる
  - サーバとかの(認証orセキュリティ)ログとかテキストからなる大規模データの加工に便利

正規表現はパズルみたいで意外と面白い

代表的な正規表現(1/2)

このあたりは他のプログラミング言語でも共通

- 文字の集合から1文字: []に文字や数字を示したもの
  - 例: [a-z], [abc], [0-9], [0-9a-zA-Z]
    - a-z, 0-9の書式で範囲を表すことが可能(マイナスを表す場合は冒頭に)
  - 冒頭に^をつけることで否定を取ることが可能 (例: [^a-z])
- 任意の1文字: .
- 0回以上の繰り返し: *「正規表現」**
- 1回以上の繰り返し: *「正規表現」*+
  - (例) 0-9の1回以上の繰り返し: [0-9]+
  - (例) aの1回以上の繰り返し: a+
  - 最小一致をさせたい場合は+の後に?をつける(例: [a-z]+?)
- 0回もしくは1回の出現: *「正規表現」*?

代表的な正規表現(2/2)

- 文頭: ^
- 文末: $
- n回の反復: *「正規表現」*{n}
  - (例) アルファベット小文字3文字: [a-z]{3}
- mからn回の反復: *「正規表現」*{m, n}
- グループ化: (*文字列*)
  - 例: (123){3}で123123123を表す
- グループの選択: (*文字列1*|*文字列2*|...)

., ^, $などの正規表現で用いられる記号を正規表現による検索対象にする場合は、¥を前につけてエスケープ(例: ¥., ¥^, ¥$)

- ¥自身を表す場合は¥¥となる

空白や文字列などを表す正規表現

- 単語の境界: ¥b
- 数字: ¥d
- 数字以外: ¥D
- 文字(Unicode文字): ¥w
- 文字(Unicode文字)以外: ¥W
- 空白文字(スペース、タブ、など): ¥s
- 空白文字(スペース、タブ、など)以外: ¥S

Pythonでは「¥」はエスケープ文字のため、「¥」をエスケープ文字に解釈されないようにする必要がある(後述)

正規表現の例

- izmで終わる単語(前後はスペース): ¥s+[a-z]+izm¥s+
- 「izm.」で終わる語(前後はスペース): ¥s+[a-z]+izm¥.¥s+
  - ピリオドはエスケープしないと「任意の1文字」になってしまう点に注意
- room0からroom9: room[0-9]
- アルファベット数文字の後の1桁の数字: [a-z]+[0-9]
  - アルファベット数文字の後の0-1桁の数字: [a-z]+[0-9]?

- 正規表現の個人的な印象
  - パズルみたいで面白い
  - 簡単なパスワードをあぶり出す正規表現も簡単に作れて便利
  - (最長一致でよくハマる)

文字列への正規表現の処理の適用 (1/2)

- 正規表現モジュール(reモジュール = Regular Expressionモジュール)を読み込む必要がある
  - プログラムの前の方で「import re」の1行を入れる
- 文字列のどこかにマッチ: re.search(*正規表現, 文字列*)
  - マッチオブジェクトを返す
  - 見つからなければNone
  - 文字列の先頭にマッチ: re.match(*正規表現, 文字列*)
    - 「文字列の先頭から」という制限以外はre.searchと同じ
    - re.searchで正規表現に^(文頭)を追加すればいいので使う意義少ない
- 文字列を正規表現のパターンで分割: re.split(*正規表現, 文字列*)
  - 正規表現で分割された物の文字列リストを返す

マッチオブジェクト

- re.search, (re.match), re.finditerでマッチがあった時にできるオブジェクト
  - マッチがないとマッチオブジェクトができず、操作しようとエラーになるので、「if *マッチオブジェクト*:」の中で操作した方が良い
- *マッチオブジェクト*.group()でマッチした内容を取り出せる
  - 後述する「()」を利用することで、*マッチオブジェクト*.group(1)などでマッチ結果の一部を参照可能
  - Python以外の言語の利用も考えると、「正規表現でマッチ→マッチした内容を取り出す」処理に慣れておくのが重要
- *マッチオブジェクト*.start()でマッチ部の先頭のインデクス(文字列のインデクス = 何文字目か)を返す
- *マッチオブジェクト*.end()でマッチ部の末尾のインデクス
- *マッチオブジェクト*.span()でマッチ部の(start, end)のタプル

文字列への正規表現の処理の適用 (2/2)

- マッチする物全て: re.findall(*正規表現*, *文字列*)
  - マッチした物の文字列リストで返す
- 置換: re.sub(*正規表現1*, *正規表現2*, *文字列*)
  - *正規表現1*を*正規表現2*で置換
- マッチする物全てをマッチオブジェクトで返すイテレータ: re.finditer(*正規表現*, *文字列*)
  - for文に対してrange関数と同じ形で使う
    - 例: for i in re.finditer()
  - ただし、個々のイテレータがマッチオブジェクトになるので、多少めんどくさいと感じることも
    - re.findallの方がお手軽

正規表現でマッチした内容の「さらに一部」を切り出す

- 正規表現の中に「(正規表現)」の部分を作ることで、そこを*マッチオブジェクト*.group(*n*)で参照可能
  - 最初の「(正規表現)」が*n*=1、2個目の「(正規表現)」が*n*=2、...
- 例: m = re.search("([0-9]+):([0-9]+):([0-9]+)", "23:38:05")
  - m.group(1) = 23
  - m.group(2) = 38
  - m.group(3) = 05
  - m.groups() = [23, 38, 05]
- マッチオブジェクト.(start|end|span)も同様にnを利用可能
  - m.start(1), m.end(2), m.span(3)

正規表現のTips

- 最長一致と最短一致
  - 何も指定しない場合には最長一致となる
    - 例:「aaab」に対して「a+b」の正規表現でマッチさせると「aaab」としてマッチする
  - 最短一致にする場合には繰り返しを示す正規表現の後に「?」を追加
    - 例:「aaab」に対して「a+?b」の正規表現でマッチさせると「ab」としてマッチする
- 正規表現中の記号がPythonに解釈されないように、正規表現はraw文字列として明示することも →今はまず不要
  - 文字列のクォーテーションの前にrをつけたりや文字列変数をrepr関数に渡すことで対応
    - 例(文字列): re.search(r"[0-9]+", "shimada1234")
    - 例(文字列変数): re.search(repr(variable_strings), "shimada1234")
  - Pythonからのエスケープ(raw文字列)と正規表現の中でのエスケープ(¥記号)の2つのエスケープが入ることがあるので混乱しないように