Python: Reguläre Ausdrücke Extended Format (C) 2014-2021 T.Birnthaler OSTC GmbH

Das Extended Format erlaubt die Formatierung und Einrückung von Regulären Ausdrücken. Das wesentliche Problem von Regex ist damit von Tisch --- dass sie so unleserlich sind, weil sie am Stück in einer Zeile zu schreiben sind.

* Grundsätzlich in Python reguläre Ausdrücke in R"""- oder R'''-Paar setzen:
  + R --> RAW/REGEX: Escape-Sequenz "\X" nicht als Sonderzeichen interpretiert
  + """ oder ''' --> Aufteilen auf MEHRERE Zeilen erlaubt

* re.VERBOSE oder re.X als Flag (3. Parameter) eintragen (eXtended Format)
  + Alles ab # bis zum Zeilenende wird ignoriert (zählt als KOMMENTAR)!
    --> Zu matchendes Zeichen "#" schützen: \# oder [#]
  + WHITESPACE (Space, Tab, VertTab, Newline, Return, FormFeed) wird ignoriert
    --> Leerraum darf zum Einrücken und Formatieren benutzt werden
    --> Zu matchendes Leerzeichen " " schützen: \ oder [ ]
    --> Zu matchenden Whitespace "..." so schreiben: \s
  + Nach "(" einrücken, vor ")" ausrücken --> Merken + Gruppieren übersichtlich
  + Analog "(?: ...) --> Nur Gruppieren (NICHT Merken)
    ACHTUNG: Syntaxfehler: (:? ...) falsch!

Beispiel:

```
  # Klassischer Regulärer Ausdruck (JEDES einzelne Zeichen hat eine Bedeutung)
  mo = re.search(r"^\s+      #(.*)\s+\1(?:.*)\s*$", line)
  #                   ^^^^^^ 6 Leerzeichen!

  # Gleicher Regulärer Ausdruck im Extended-Format (formatiert + kommentiert):
  mo = re.search(        # MO=MatchObject
       r"""              # raw/regex + Zeilenumbruch erlaubt
           ^             # Zeilenanfang
           \s+           # 1-n Leerraum (whiteSpace)
           \ \ \         # 3 Leerzeichen
           [ ][ ][ ]     # 3 Leerzeichen
           \#            # Zeichen "#"!
           (             # Merken 1: Einrücken
               .*        # Merken 1: 0-n beliebige Zeichen
           )             # Merken 1: Ausrücken
           \s+           # 1-n Leerraum (whiteSpace)
           \1            # Text bei "Merken 1" nochmal!
           (?: .* )      # Klammern, aber nicht merken (z.B. weil optional)
           \s*           # 0-n Leerraum (whiteSpace)
           $             # Zeilenende
       """, line, re.VERBOSE)   # re.X = re.VERBOSE = Extended Format
  if (mo):
      print "Matchteile:          ", len(mo.groups())
      print "Vollständiger Match:",  mo.group(0)
      print "Teilmatch:           ", mo.group(1)
  else:
      print "KEIN match"
```

TIPPS
-----
* R"..." oder R'...' bzw. R"""...""" oder R'''...''' verwenden (RAW/REGEX)
  --> Metazeichen "\" (ESCAPE) bleibt erhalten

* Öffnende Merkklammern "(" von links nach rechts durchnumeriert von 1 .. n
  --> Mit \1 \2 ... \n kann auf davon gematchten Inhalt zugegriffen werden
      (in re.search, re.match, re.sub(...))

* mo.groups() enthält Liste von gemerkten Textstücke (inkl. leere = None!)
  --> Klammern im Regex kennzeichnen die zu merkenden Textstücke
  --> Optionale Teile enthalten "None", wenn es keinen Treffer gab
  --> len(mo.groups()) ist Anzahl gemerkter Textstücke
  + mo.group(0) enthält GANZEN gematchten Text (nicht unbedingt ganze Zeile)
  + mo.group(1) enthält 1. gemerkten Textteil (1. Klammer (...))
  + mo.group(2) enthält 2. gemerkten Textteil (2. Klammer (...))
  + ...
  mo.lastindex enthält letzte Gruppennummer

* Zu jeder Gruppe mo.group(N) gibt es folgende weiteren Informationen:
    mo.start(N)   Start-Index der Gruppe
    mo.end(N)   End-Index der Gruppe
    mo.span(N)   Tupel (Start, End)-Index der Gruppe
    mo.len(N)   Laenge der Gruppe

```
* re.match() und re.fullmatch() NICHT verwenden, sind Teil von re.search()
  + re.match("...")     = re.search("^...")       # Ab Zeilenanfang
  + re.fullmatch("...") = re.search("^...$")      # Vollständig

* Reguläre Ausdrücke sind "greedy"
  --> Suchen längsten Treffer
  --> .* alleine ist sinnlos (passt auf alles und nichts),
      es sollte immer etwas aussenrum stehen
  --> .*? ist "non-greedy" oder "lazy" und matcht kürzestmöglichen Text

* Reguläre Ausdrücke sind "left-most"
  --> Suchen von links nach rechts den ersten Treffer

* Erst "Schmutz" wegwerfen, dann Verarbeitung per Regex
  + Z.B. Leerzeilen und Kommentarzeilen ignorieren per:

        if re.search(r"^\s*[#]?\s*$", line): continue

  + Whitespace am Zeilenanfang oder Ende entfernen
    (inkl. Newline, Carriage Return, Tab, ...)

        line = re.sub(r"^\s+", "", line)
        line = re.sub(r"\s+$", "", line)

* Statt .* besser [^FOLGEZEICHEN]* schreiben
  --> "greedy"-Verhalten wird gestoppt

* Ein "." matcht ALLE Zeichen AUSSER "\n" (Newline).
  Flag re.S / re.DOTALL schaltet dieses Verhalten ab (Newline auch gematcht)
  Alternative: "[\s\S]" matcht auch JEDES Zeichen

* "^" und "$" matchen String-Anfang/Ende.
  Flag re.M / re.MULTILINE sorgt dafür, dass sie auch "\n" matchen
  (für mehrzeilige Daten in einem String interessant).
  Die bisherige Aufgabe der beiden übernehmen "\A" und "\Z" bzw. "\z".
```