Reguliere expressies

Oplossingen van enkele opgaven in Spronck hoofdstuk 25

Het boek:
Pieter Spronck
De Programmeursleerling — Leren coderen met Python 3
versie 1.0.16
24 oktober 2017

Mijn vorige document:
Spronck-hoofdstuk-25-opgaven.pdf

Mijn vorige listings:
Hoofdstuk-25-listing2506-uitgesplitst--OPGAVE.py
Hoofdstuk-25-listing2510--OPGAVEN.py
Hoofdstuk-25-listing2510alt--OPGAVEN.py

Mijn nieuwe listings:
Hoofdstuk-25-listing2506-uitgesplitst--OPLOSSING.py
Hoofdstuk-25-listing2510--OPLOSSINGEN.py
Hoofdstuk-25-listing2510alt--OPLOSSINGEN.py

Inhoud

1 Een oplossing bij listing 2506

```
# Hoofdstuk-25-listing2506-uitgesplitst--OPLOSSING.py -- 9 re's voor 8 strings
# Uit Pieter Spronck: De programmeursleerling
# Variant door Meindert Meindertsma, 2023-05-22


import re

# List van strings die worden gebruikt voor testen.
slist = [ "aaabbb", "aaaaaa", "abbaba", "aaa", "bEver ottEr",
"tango samba rumba", " hello world ", " Hello World " ]

# Reguliere expressies die moeten worden ingevuld.
relist = [
    None,
    r"^a+b*$",            # 1. Alleen a's gevolgd door alleen b's, inclusief ""
    r"^a*$",              # 2. Alleen a's, inclusief ""
    r"^[ab]*$",           # 3. Alleen a's en b's, willekeurige volgorde, incl. ""
    r"^aaa$",             # 4. Precies drie a's
    r"^[^ab]*$",          # 5. Noch a's noch b's, maar "" is toegestaan
    r"^(aa)*$",           # 6. Een even aantal a's (en niks anders)
    r"^\W*\w+\W+\w+\W*$", # 7. Precies twee woorden, ongeacht spaties
    r"ba\b",              # 8. Bevat een woord dat op "ba" eindigt
    r"\b[A-Z]"            # 9. Bevat een woord dat begint met een hoofdletter
]

# Uitgesplitste presentatie.
for i in range(1, len( relist ) ):
# for i in (1,):
    print( f'Reguliere expressie {i} = "{relist[i]}":' )
    for s in slist:
        if m := re.search( relist[i], s ):
            gevonden = f'segment [{m.start(0)}:{m.end(0)}] = "{m.group(0)}"' \
                       f' voldoet aan re-{i} "{relist[i]}"'
        else:
            gevonden = ""
        print( f'''   {f'"{s}"' :19s}:''', gevonden )
    print()
```

De uitvoer hiervan:

```
Reguliere expressie 1 = "^a+b*$":
   "aaabbb"           : segment [0:6] = "aaabbb" voldoet aan re-1 "^a+b*$"
   "aaaaaa"           : segment [0:6] = "aaaaaa" voldoet aan re-1 "^a+b*$"
   "abbaba"           :
   "aaa"              : segment [0:3] = "aaa" voldoet aan re-1 "^a+b*$"
   "bEver ottEr"      :
   "tango samba rumba":
   " hello world "    :
   " Hello World "    :

Reguliere expressie 2 = "^a*$":
   "aaabbb"           :
   "aaaaaa"           : segment [0:6] = "aaaaaa" voldoet aan re-2 "^a*$"
   "abbaba"           :
   "aaa"              : segment [0:3] = "aaa" voldoet aan re-2 "^a*$"
   "bEver ottEr"      :
   "tango samba rumba":
   " hello world "    :
   " Hello World "    :
```

```
Reguliere expressie 3 = "^[ab]*$":
   "aaabbb"           : segment [0:6] = "aaabbb" voldoet aan re-3 "^[ab]*$"
   "aaaaaa"           : segment [0:6] = "aaaaaa" voldoet aan re-3 "^[ab]*$"
   "abbaba"           : segment [0:6] = "abbaba" voldoet aan re-3 "^[ab]*$"
   "aaa"              : segment [0:3] = "aaa" voldoet aan re-3 "^[ab]*$"
   "bEver ottEr"      :
   "tango samba rumba":
   " hello world "    :
   " Hello World "    :

Reguliere expressie 4 = "^aaa$":
   "aaabbb"           :
   "aaaaaa"           :
   "abbaba"           :
   "aaa"              : segment [0:3] = "aaa" voldoet aan re-4 "^aaa$"
   "bEver ottEr"      :
   "tango samba rumba":
   " hello world "    :
   " Hello World "    :

Reguliere expressie 5 = "^[^ab]*$":
   "aaabbb"           :
   "aaaaaa"           :
   "abbaba"           :
   "aaa"              :
   "bEver ottEr"      :
   "tango samba rumba":
   " hello world "    : segment [0:13] = " hello world " voldoet aan re-5 "^[^ab]*$"
   " Hello World "    : segment [0:13] = " Hello World " voldoet aan re-5 "^[^ab]*$"

Reguliere expressie 6 = "^(aa)*$":
   "aaabbb"           :
   "aaaaaa"           : segment [0:6] = "aaaaaa" voldoet aan re-6 "^(aa)*$"
   "abbaba"           :
   "aaa"              :
   "bEver ottEr"      :
   "tango samba rumba":
   " hello world "    :
   " Hello World "    :

Reguliere expressie 7 = "^\W*\w+\W+\w+\W*$":
   "aaabbb"           :
   "aaaaaa"           :
   "abbaba"           :
   "aaa"              :
   "bEver ottEr"      : segment [0:11] = "bEver ottEr" voldoet aan re-7 "^\W*\w+\W+\w+\W*$"
   "tango samba rumba":
   " hello world "    : segment [0:13] = " hello world " voldoet aan re-7 "^\W*\w+\W+\w+\W*$"
   " Hello World "    : segment [0:13] = " Hello World " voldoet aan re-7 "^\W*\w+\W+\w+\W*$"

Reguliere expressie 8 = "ba\b":
   "aaabbb"           :
   "aaaaaa"           :
   "abbaba"           : segment [4:6] = "ba" voldoet aan re-8 "ba\b"
   "aaa"              :
   "bEver ottEr"      :
   "tango samba rumba": segment [9:11] = "ba" voldoet aan re-8 "ba\b"
   " hello world "    :
   " Hello World "    :

Reguliere expressie 9 = "\b[A-Z]":
   "aaabbb"           :
   "aaaaaa"           :
   "abbaba"           :
   "aaa"              :
   "bEver ottEr"      :
   "tango samba rumba":
   " hello world "    :
   " Hello World "    : segment [1:2] = "H" voldoet aan re-9 "\b[A-Z]"
```

De reviewer van mijn stukken vond dat de alternatieve interpretatie van de eerste reguliere expressie niet onderdeed voor die welke ik prefereerde. Beide goed, reageerde ik. Daarom beide uitwerkingen hier expliciet vergeleken:

- Nauwe interpretatie `^a+b*$`: “aaa” voldoet, “” niet.
- Ruime interpretatie `^a*b*$`: “aaa” voldoet, “” eveneens.

2 Oplossingen bij listing 2510

```
# Hoofdstuk-25-listing2510--OPLOSSINGEN.py -- 2 opgaven
# Uit Pieter Spronck: De programmeursleerling
# Variant door Meindert Meindertsma, 2023-05-22


import re

# List van strings die worden gebruikt voor testen.
slist = [ "Monty Python's Flying Circus", "aaaa", "aabb", "abab", "abba" ]

# Reguliere expressies die moeten worden ingevuld.
relist = [
    None,
    r"(.).*\1.*\1",      # 1. Willekeurig teken dat 3 keer voorkomt
    r"(..).*\1",         # 2. Willekeurig tekenpaar dat 2 keer voorkomt
    r"((.)(?!\2).).*\1"  # 3. Willekeurig verschillend tekenpaar
                         #    dat 2 keer voorkomt
]

# Uitgesplitste presentatie.
for i in range(1, len( relist ) ):
# for i in (1,):
    print( f'Reguliere expressie {i} = "{relist[i]}":' )
    for s in slist:
        if m := re.search( relist[i], s ):
            gevonden = f'segment [{m.start(0)}:{m.end(0)}] = "{m.group(0)}"' \
                       f' voldoet aan re-{i} "{relist[i]}"'
        else:
            gevonden = ""
        print( f'''   {f'"{s}"' :19s}:''', gevonden )
    print()
```

De bijbehorende uitvoer:

```
Reguliere expressie 1 = "(.).*\1.*\1":
   "Monty Python's Flying Circus": segment [2:20] = "nty Python's Flyin" voldoet
   "aaaa"             : segment [0:4] = "aaaa" voldoet
   "aabb"             :
   "abab"             :
   "abba"             :

Reguliere expressie 2 = "(..).*\1":
   "Monty Python's Flying Circus": segment [1:12] = "onty Python" voldoet
   "aaaa"             : segment [0:4] = "aaaa" voldoet
   "aabb"             :
   "abab"             : segment [0:4] = "abab" voldoet
   "abba"             :

Reguliere expressie 3 = "((.)(?!\2).).*\1":
   "Monty Python's Flying Circus": segment [1:12] = "onty Python" voldoet
   "aaaa"             :
   "aabb"             :
   "abab"             : segment [0:4] = "abab" voldoet
   "abba"             :
```

In de laatste reguliere expressie zag ik me echt genoopt een mechanisme te gebruiken dat Spronck niet behandelt en ik zelf maar hoogst, hoogst zelden toepas: de *zero-width lookahead assertion*. 'Zero-width' houdt in dat een match niet wordt 'geconsumeerd'. Er bestaat een *zero-width positive lookahead assertion* `(?=.....)` en een *zero-width negative lookahead assertion* `(?!.....)`. Ik heb de negatieve vorm `((.)(?!\2).*\1` gebruikt. Wanneer we de uitdrukking tussen de buitenste haakjes, dus `(.)(?!\2)`, even vervangen door *patroon1*, kunnen we de grote lijnen herkennen:

- `(patroon1).*\1` betekent: *patroon1*, gevolgd door zo veel mogelijk willekeurige tekens, gevolgd door de substring die door *patroon1* gevonden was.
- *patroon1* staat voor `(patroon2)(?!\2).` en betekent: *patroon2* mits vooruitblikkend de hierop aansluitende substring niet gelijk blijkt aan de substring die door *patroon2* gevonden was, en dan direct op *patroon2* volgend een willekeurig teken.
- *patroon2* staat voor `.` en betekent: een willekeurig teken.

Op die manier voldoet "aa" niet aan `(.)(?!\2).` omdat de tweede letter gelijk is aan de eerste, en "ab" wel omdat de tweede letter verschilt van de eerste en de eerste volgens de laatste punt wordt gevolgd door een willekeurig teken. Die punt heeft dus niet betrekking op een derde teken, maar op het tweede!

Overigens werkt de volgende *zero-width negative lookbehind* `((.).(?<!\2)).*\1` hier ook, d.w.z. met hetzelfde resultaat. Het is nu bij het schrijven van deze alinea dat ik die vorm voor het eerst van mijn leven gebruik. Vermoedelijk ook voor het laatst.

In eerste instantie hoopte ik met `((.)[^\2]).*\1` iets te bereiken. De `\2` in een reguliere expressie tussen vierkante haken oftewel 'tekenklasse' (Engelse vakterm: *character class*) betekent echter iets heel anders dan daarbuiten. Erbuiten is het een verwijzing naar een gevonden substring in groep 2. Erbinnen is het een verwijzing naar één specifiek teken. In dit geval het teken met volgnummer 2 in de ASCII- of Unicode-tabel (derde rij in de kolom met kop `0.`):

	0.	1.	2.	3.	4.	5.	6.	7.
.0	0 ^@ NUL	16 ► ^P DLE	32	48 0	64 @	80 P	96 `	112 p
.1	1 ☺ ^A SOH	17 ◄ ^Q DC1	33 !	49 1	65 A	81 Q	97 a	113 q
.2	2 ☻ ^B STX	18 ↕ ^R DC2	34 "	50 2	66 B	82 R	98 b	114 r
.3	3 ♥ ^C ETX	19 ‼ ^S DC3	35 #	51 3	67 C	83 S	99 c	115 s
.4	4 ♦ ^D EOT	20 ¶ ^T DC4	36 $	52 4	68 D	84 T	100 d	116 t
.5	5 ♣ ^E ENQ	21 § ^U NAK	37 %	53 5	69 E	85 U	101 e	117 u
.6	6 ♠ ^F ACK	22 ▬ ^V SYN	38 &	54 6	70 F	86 V	102 f	118 v
.7	7 • ^G BEL	23 ↨ ^W ETB	39 '	55 7	71 G	87 W	103 g	119 w
.8	8 ◘ ^H BS	24 ↑ ^X CAN	40 (	56 8	72 H	88 X	104 h	120 x
.9	9 ○ ^I HT	25 ↓ ^Y EM	41)	57 9	73 I	89 Y	105 i	121 y
.A	10 ◙ ^J LF	26 → ^Z SUB	42 *	58 :	74 J	90 Z	106 j	122 z
.B	11 ♂ ^K VT	27 ← ^[ESC	43 +	59 ;	75 K	91 [	107 k	123 {
.C	12 ♀ ^L FF	28 ∟ ^\ FS	44 ,	60 <	76 L	92 \	108 l	124 ¦
.D	13 ♪ ^M CR	29 ↔ ^] GS	45 -	61 =	77 M	93]	109 m	125 }
.E	14 ♫ ^N SO	30 ▲ ^^ RS	46 .	62 >	78 N	94 ^	110 n	126 ~
.F	15 ☼ ^O SI	31 ▼ ^_ US	47 /	63 ?	79 O	95 _	111 o	127 ⌂

Wat als ik het met benoemde groepen `((?P<twee>.)[^(?P=twee)]).*\1` zou proberen (zie Spronck blz. 299)? Helaas, dat werkt ook niet. De referentie `(?P=twee)` naar de tekenklasse `twee` (jammer genoeg weer niet in Spronck beschreven) wordt nu letterlijk genomen. In dit geval dus: de in groep `twee` gevonden substring mag niet gelijk zijn aan een van de tekens `(`, `?`, `P`, `=`, `t`, `w`, `e`, of `)`. Buiten de tekenklasse werkt zo'n referentie wel naar behoren. Zo zou "aa" prima voldoen aan `(?P<twee>.)(?P=twee)`, maar dus niet aan `(?P<twee>.)[(?P=twee)]`.

3 Oplossingen bij andere aanpak van listing 2510

Oorspronkelijk had ik de opgaven bij listing 2510 op een heel andere manier uitgewerkt, namelijk door alle suggesties van Spronck in één lange teststring onder te brengen:

> "Monty Python's Flying Circus aaaa aabb abab abba"

Daarbij zag ik oplossingen verschijnen die zich over verschillende teststrings uitstrekten, wat waarschijnlijk niet de bedoeling van Spronck was. Maar het bracht me wel op het idee om ook rekening te houden met spaties en om het fenomeen *non-greedy search* onder de aandacht te brengen. Dat laatste is m.i. een onontbeerlijk stukje gereedschap bij reguliere expressies. Ik pas het vaak toe, vooral in de combinatie **`*?`**. Zie verder de tweede en vierde `verwerk`-aanroep t.b.v. opgave 2:

```
# Hoofdstuk-25-listing2510alt--OPLOSSINGEN.py -- 2 opgaven
# Uit Pieter Spronck: De programmeursleerling
# Oplossingen door Meindert Meindertsma, 2023-05-12


import re

def verwerk( titel, patroon ):
    print( "   ", titel, f'-- patroon = "{patroon}":')
    for m in re.finditer( patroon, tekst ):
        print( "       ", m, f'"{m.group(1)}"' if len(m.groups()) > 0 else '' )

#------------------------------------------------------------------------------#
"""
Opgave: Kun je de reguliere expressie in de code hieronder zo wijzigen dat het een
patroon beschrijft waarbij een willekeurig teken minimaal drie keer voorkomt?
"""
tekst = "Monty Python's Flying Circus"
print( f'Tekst = "{tekst}"' )

verwerk( "1e opgave", r"(.).*?\1.*?\1" )

#------------------------------------------------------------------------------#
"""
Opgave: Kun je de reguliere expressie wijzigen zodat het een patroon beschrijft waarbij
twee tekens twee keer voorkomen? Dit is behoorlijk moeilijk en daarom optioneel, maar
als je het probeert, zorg er dan voor dat je het test met op zijn minst de strings "aaaa",
"aabb", "abab" and "abba". Deze moeten allemaal een match geven, tenzij je ook nog eist
dat de twee tekens verschillend moeten zijn, in welk geval "aaaa" geen match mag geven
(maar dat maakt de reguliere expressie nog eens een stuk moeilijker).
"""
tekst += " aaaa aabb abab abba"
print( f'\nTekst = "{tekst}"' )

verwerk( "2e opgave, GREEDY, willek. non-spatiepaar"        , r"(\S\S).*\1" )

verwerk( "2e opgave, idem NON-GREEDY"                       , r"(\S\S).*?\1" )

verwerk( "2e opgave, GREEDY, willek. versch. non-spatiepaar", r"((\S)(?!\2)\S).*\1" )

verwerk( "2e opgave, idem NON-GREEDY"                       , r"((\S)(?!\2)\S).*?\1" )
```

De bijbehorende uitvoer:

```
Tekst = "Monty Python's Flying Circus"
    1e opgave -- patroon = "(.).*?\1.*?\1":
        <re.Match object; span=(2, 20), match="nty Python's Flyin"> "n"

Tekst = "Monty Python's Flying Circus + aaaa aabb abab abba"
    2e opgave, GREEDY, willek. non-spatiepaar -- patroon = "(\S\S).*\1":
        <re.Match object; span=(1, 12), match='onty Python'> "on"
        <re.Match object; span=(29, 36), match='aaaa aa'> "aa"
        <re.Match object; span=(36, 47), match='bb abab abb'> "bb"
    2e opgave, idem NON-GREEDY -- patroon = "(\S\S).*?\1":
        <re.Match object; span=(1, 12), match='onty Python'> "on"
        <re.Match object; span=(29, 33), match='aaaa'> "aa"
        <re.Match object; span=(35, 41), match='abb ab'> "ab"
        <re.Match object; span=(41, 46), match='ab ab'> "ab"
    2e opgave, GREEDY, willek. versch. non-spatiepaar -- patroon = "((\S)(?!\2)\S).*\1":
        <re.Match object; span=(1, 12), match='onty Python'> "on"
        <re.Match object; span=(35, 46), match='abb abab ab'> "ab"
    2e opgave, idem NON-GREEDY -- patroon = "((\S)(?!\2)\S).*?\1":
        <re.Match object; span=(1, 12), match='onty Python'> "on"
        <re.Match object; span=(35, 41), match='abb ab'> "ab"
        <re.Match object; span=(41, 46), match='ab ab'> "ab"
```

4 Listings kopiëren?

Knippen en plakken van tekst uit een PDF-document blijkt een hachelijke zaak, vooral waar het om Python-code gaat. Daarom lever ik de drie programmalistings die in dit document zijn afgedrukt (zie ook pagina 1) als aparte bestanden mee.