Reguliere expressies

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

Inhoud

1 Inleiding

Hoofdstuk 25 bevat enige nogal pittige opgaven. In sommige gevallen is het bovendien nog eens onduidelijk wat er precies van de zelf op te stellen reguliere expressies verwacht wordt. Daarom kleed ik in dit document drie als struikelblok ervaren vraagstukken nog wat verder aan, in de hoop alle bijkomende verwarring op die manier weg te nemen. Het blijven evengoed pittige opgaven, dat wel.

2 Opgave bij listing 2506 op bladzijde 297 en 298

```
# Hoofdstuk-25-listing2506-uitgesplitst--OPGAVE.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"",  # 1. Alleen a's gevolgd door alleen b's, inclusief ""
    r"",  # 2. Alleen a's, inclusief ""
    r"",  # 3. Alleen a's en b's, willekeurige volgorde, incl. ""
    r"",  # 4. Precies drie a's
    r"",  # 5. Noch a's noch b's, maar "" is toegestaan
    r"",  # 6. Een even aantal a's (en niks anders)
    r"",  # 7. Precies twee woorden, ongeacht spaties
    r"",  # 8. Bevat een woord dat op "ba" eindigt
    r""   # 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 zoals die op bladzijde 298 van het Pythonboek staat, wordt keer op keer verkeerd geïnterpreteerd, ook door mij. Daarom heb ik het programma een beetje aangepast. Voor elke reguliere expressie wordt nu een apart overzichtje voor de acht strings in `slist` afgedrukt.

> We kunnen, als we daar de tijd voor en zin in hebben, nu even stilstaan bij een ergonomische of anders wel esthetische kwestie. Spronck liet de reguliere-expressieteller van `0` t/m `8` lopen, maar wilde ze als `1` t/m `9` presenteren. Vandaar dat hij bij elke `print` nog eens `1` bij de index optelt. Geen afleidingsmanoeuvre, maar desalniettemin een constructie die afleidt van waar het in de opgave om gaat. In mijn vervangende code heb ik die extra operatie weten weg te werken door een dummy met index `0` aan het begin van `relist` toe te voegen. Daar valt natuurlijk net zo goed wel iets op af te dingen.

Voor bijvoorbeeld de eerste reguliere expressie ziet zo'n overzichtje er aldus uit:

```
Reguliere expressie 1 = "........":
   "aaabbb"           : segment [........] = "........" voldoet aan re-1 "........"
   "aaaaaa"           : segment [........] = "........" voldoet aan re-1 "........"
   "abbaba"           :
   "aaa"              : segment [........] = "........" voldoet aan re-1 "........"
   "bEver ottEr"      :
   "tango samba rumba":
   " hello world "    :
   " Hello World "    :
```

Dat correspondeert met alle enen in de uitvoer van het oorspronkelijke programma:

```
aaabbb  :  1 3
aaaaaa  :  1 2 3 6
abbaba  :  3 8
aaa  :  1 2 3 4
bEver ottEr  :  7
tango samba rumba  :  8
 hello world   :  5 7
 Hello World   :  5 7 9
```

In mijn programmavariant op de vorige pagina heb ik `for i in range(1, len(relist)):` uitgecommentarieerd en vervangen door `for i in (1,):`. Door steeds het nummer van een reguliere expressie waaraan je wilt werken toe te voegen aan de tuple kun je de opgave stap voor stap verder uitwerken en de uitvoer daarbij overzichtelijk houden. Je kunt bovendien met een willekeurige reguliere expressie beginnen.

Een aantal bijschriften bij de reguliere expressies vond ik lastig te duiden. Uiteindelijk ben ik na enig experimenteren tot de volgende betekenissen gekomen:

1. De string mag alleen bestaan uit a's gevolgd door alleen b's of door helemaal niets. Een lege string is misschien ook wel goed (komt in deze opgave toevallig niet voor). De string "abbaba" voldoet niet, omdat er na de eerste a nog iets anders volgt dan alleen maar b's.
2. De string mag niets anders dan a's bevatten. Een lege string is dus ook goed.
3. De string mag niets anders dan a's en/of b's bevatten.
5. Zolang er maar geen a's en geen b's in de string zitten, is het in orde.
6. Is nul een even aantal? Ik vind van wel. Modulo `2` levert `0` op, net als bij alle andere even natuurlijke getallen. Maar even of niet, de laatste twee strings (" hello world " en " Hello World ") voldoen sowieso niet.
7. Ik houd het er maar op dat alle spaties en interpunctie buiten een woord vallen.

Vaak kun je verschillende reguliere expressies opstellen die precies hetzelfde effect hebben. Niets let je dus om na negen invullingen nog even door te gaan, als je zin hebt.

3 Opgaven bij listing 2510 op bladzijde 300

Het programma in listing 2510 heb ik drastisch omgebouwd door hetzelfde model toe te passen als ik bij listing 2506 heb gedaan. Er moeten drie reguliere expressiesvoor vijf teststrings worden ingevuld:

```
# 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. Willekeurig teken dat 3 keer voorkomt
    r"",  # 2. Willekeurig tekenpaar dat 2 keer voorkomt
    r""   # 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()
```

Voor opgave 2 moeten dus twee reguliere expressies worden ingevuld:

(a) Reguliere expressie 2 voor de ‘gemakkelijke’ variant: maak een patroon voor paren van twee willekeurige tekens die twee keer *als zo’n zelfde paar in dezelfde volgorde* voorkomen (mijn toegevoegde clausule gecursiveerd).

(b) Reguliere expressie 3 voor twee keer voorkomende paren van twee onderling verschillende tekens. Het paar “aa” kan dus nooit voldoen, het paar “ab” wel.

4 Alternatieve opgaven bij listing 2510

Aanvankelijk had ik een heel ander programma geschreven. Ten opzichte van de originele listing 2510 had ik het volgende veranderd:

- De regel `m = re.search( r"(\S).*\1", "Monty Python's Flying Circus" )` is vervangen door de aanroep van een tevoren gedefinieerde functie `verwerk`.
- Deze functie roept niet `re.search( patroon, tekst)` aan, maar de iterator `re.finditer`: `re.finditer( patroon, tekst )`.
- Voor elke match `m` die deze iterator vindt, wordt deze match afgedrukt, samen met de waarde van de eerste groep. Voor het geval dat je geen gebruik maakt van groepen, heb ik de laatste parameter van print omgewerkt tot een *ternary if/else expression* zoals ook voorkomt in

listing 2203 op bladzijde 272 en op de daaraan voorafgaande pagina wordt uitgelegd: `value1 if condition else value2`.

Voor de eerste opgave had ik het bij de tekst “Monty Python's Flying Circus” gelaten. Voor de tweede had ik deze naar aanleiding van Sproncks advies aangevuld met “ aaaa aabb abab abba”. Helemaal in overeenstemming met zijn bedoeling was dit waarschijnlijk niet, want op deze manier kun je oplossingen krijgen die zich over meer dan één teststring uitstrekken. Wanneer je er nog voor voelt, kun je ook nog eens met deze alternatieve opgaven aan de slag. Door die lange teststring zitten er wel aardige dingen in, heb ik gemerkt. Daarom heb ik de tweede opgave een beetje veranderd:

```
# Hoofdstuk-25-listing2510alt--OPGAVEN.py -- 2 opgaven
# Uit Pieter Spronck: De programmeursleerling
# Variant door Meindert Meindertsma, 2023-05-23


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"" )

#-------------------------------------------------------------------------------#
"""
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. tekenpaar ex witruimte", r"" )

verwerk( "2e opgave, idem NON-GREEDY"                        , r"" )

verwerk( "2e opgave, GREEDY, willek. verschillend tekenpaar", r"" )

verwerk( "2e opgave, idem NON-GREEDY"                        , r"" )
```

De eerste opgave is precies hetzelfde gebleven. De tweede heb ik uitgesplitst in vieren:

(a) Willekeurige tekenparen die twee keer voorkomen, maar nu met de restrictie dat *spaties en andere witruimte (tabs, carriage returns, line feeds) geen deel van zo’n paar mogen uitmaken*. Witruimte *tussen* de twee gevonden tekenparen is wel toegestaan.

(b) Als *(a)*, met als extra eis dat de vondsten zo kort mogelijk moeten zijn (met als mogelijk bijeffect dat je meer van zulke paren vindt.

(c) Als *(a)*, maar met de extra restrictie dat de twee tekens in zo'n paar onderling verschillend moeten zijn.
(d) Als *(c)*, met ook weer zo kort mogelijke matches.

Voor *(b)* en *(d)* is nog wel een trucje nodig dat Spronck ons niet geleerd heeft. Op bladzijde 296 merkt hij op dat het matchen van herhalingen 'gulzig' gebeurt. Die gulzigheid valt echter te temperen door een vraagteken achter de herhalingsoperator te zetten. Dan geschiedt het matchen namelijk juist heel 'terughoudend' (Engels: *non-greedy* / *lazy* / *reluctant* / *moderate* / *abstemious*). In de string "aabbcc" vindt de reguliere expressie `a.*b` de substring "aabb", terwijl `a.*?b` het bij "aab" (niet "ab"!) laat.

Mijn waarschuwing in de vorige alinea geeft aan dat 'zo kort mogelijk' niet hetzelfde is als 'zo kort als men zich maar kan voorstellen'. Ik denk namelijk dat dat laatste met reguliere expressies alleen niet te bereiken is. Voor de echte diehards wellicht een interessante uitdaging om mijn vermoeden te weerleggen.

De uitvoer heeft onderstaande structuur:

```
Tekst = "Monty Python's Flying Circus"
    1e opgave -- patroon = ".........":
        <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 = ".........":
        <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 = ".........":
        <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 = ".........":
        <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 = ".........":
        <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"
```

De reguliere expressies die ik hierbij heb toegepast, heb ik weliswaar uitgegumd, maar de matches niet. De lezer mag proberen de juiste patronen te bedenken die tot deze uitvoer leidt. Ook hier geldt dat er per geval meer oplossingen goed zouden kunnen zijn. Of misschien zelfs tot betere resultaten c.q. andere uitvoer zouden leiden dan de mijne (ik verwacht het niet).

5 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.

6 Oplossingen

De oplossingen die ik gevonden heb, zal ik publiceren in een apart document met bestandsnaam "Spronck-hoofdstuk-25-oplossingen.pdf".

7 Commentaar

Hieronder sta ik stil bij enige opmerkingen die een kritische lezer van een conceptversie van dit document maakte. Ook nuttig voor andere lezers.

1. Mijn interpretatie van de eis die aan de eerste reguliere expressie in listing 2506 is gesteld houdt in dat een lege string niet voldoet, maar een string met alleen maar a's wel. Een andere mogelijke interpretatie is, dat een geheel lege string ook voldoet. Ik reken beide interpretaties goed. Misschien een leuk idee om beide interpretaties daadwerkelijk uit te proberen. Voeg daartoe een lege string aan `slist` toe en een tiende reguliere expressie aan `relist` (misschien vooraan, als element met index `0`?).
2. In mijn listing 2506 komt een uitgecommentarieerde `for`-lus voor. Als juist deze vorm wordt gekozen, dan is er wel wat voor te zeggen om de index **`i`** alleen te gebruiken om af te drukken en de bijbehorende waarde rechtstreeks uit `relist` op halen door de `enumerate`-functie te gebruiken. Dat maakt starten vanaf `0` misschien minder bezwaarlijk. De listing zou daarmee zo eindigen:

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

3. In mijn oorspronkelijke bewerking van listing 2506 had ik de toekenning van een lange stringwaarde aan de variabele `gevonden` uitgesplitst in twee toekenningen. De reviewer attendeerde me erop dat je een string-literal over twee regels kunt opbreken door de uitdrukking tussen haakjes te zetten (binnen de haakjes is de regelopmaak wat Python betreft geheel vrij). Dat wist ik wel, maar ik ben niet zo dol op haakjes. Wat ik niet wist, is dat je het plusteken bij concatenatie kunt weglaten. Uiteindelijk ben ik tot de huidige oplossing zonder plus en zonder haakjes, maar met een backslash gekomen. Laatstgenoemde maakt van twee fysieke regels één logische regel, dus Python zeurt niet over foute inspringing.
4. Als `gevonden` vóór het `if`-statement in 2506 wordt geïnitialiseerd met **`gevonden = ""`**, kan de `else`-tak vervallen. Dat scheelt een regel.
5. Over mijn alternatieve bewerking van listing 2510 in hoofdstuk 4: de functie `verwerk` gebruikt de globale variabele `tekst`. Gelukkig alleen maar om te lezen, maar toch niet zo netjes. Doorgifte als functieparameter ware beter.