Modul 7

Regular Expressions

Regular expression, sering disebut *regex*, adalah suatu bahasa yang handal untuk mencocokkan pola di suatu string. Praktikum ini akan memperkenalkan peserta pada fasilitas-fasilitas regex yang ada pada Python[1]. Modul Python yang memiliki fasilitas ini adalah modul `re`. Mulailah memuat modul itu ke memori dengan perintah

```
>>> import re
```

Dalam Python, regex biasa ditulis dalam bentuk

```
cocok = re.findall(pola, sebuahString)
```

Metode `re.findall()` di atas menerima sebuah pola regex dan sebuah string, lalu mencari rangkaian karakter yang sesuai dengan pola itu di string yang dimaksud. Jika pencariannya berhasil, `findall()` mengembalikan sebuah *list* berisi semua string yang ditemukan dan mengembalikan list kosong kalau tidak berhasil. Contoh berikut akan mencari pola `'kata:'` diikuti sebuah kata 3 huruf. Ketiklah lalu jalankan.

```
1 s = 'sebuah contoh kata:teh!!'
2 cocok = re.findall(r'kata:\w\w\w', s)
3 # Pernyataan-IF sesudah findall() akan memeriksa apakah pencarian berhasil:
4 if cocok:
5     print('menemukan', cocok ) ## 'menemukan [kata:teh]'
6 else:
7     print('tidak menemukan')
```

Kode `cocok = re.findall(pola, s)`, seperti pada baris 2 di atas, menyimpan hasil pencarian pada sebuah variabel bernama "cocok". Sesudah itu pernyataan-IF akan memeriksa `cocok` – jika berisi sesuatu maka pencariannya berhasil dan `cocok` adalah list teks-teks yang cocok (dalam contoh di atas adalah satu buah `'kata:teh'`). Selain itu jika cocok-nya kosong, maka berarti tidak ada teks yang cocok dengan polanya.

Huruf 'r' pada permulaan teks pola menunjukkan teks "mentah" yang melewatkan karakter *backslash* (yakni "\") tanpa perubahan. Kamu disarankan agar selalu memakai 'r' ini ketika menulis regex.

[1]Di sini kita tidak akan belajar bagaimana *membuat* mesin regex-nya. Itu merupakan topik lanjut.

7.1 Pola-Pola Dasar

Kekuatan regex adalah bahwa dia bisa membuat pola-pola khusus, bukan hanya karakter yang *fixed*. Berikut ini adalah pola-pola paling dasar yang akan cocok dengan sebuah atau beberapa karakter.

Jika saat pertama kali membaca kamu tidak paham maksud pola-pola di bawah ini, jangan putus asa! Kerjakan saja dulu latihan-latihan di modul ini, sedikit demi sedikit, lalu nanti kembali ke sini untuk referensi.

- `a, X, 9` — karakter-karakter biasa akan cocok dengan tepat. Karakter khusus yang tidak dengan sendirinya cocok, karena punya makna khusus, adalah `.^$*+?{}[]\|()` (karakter ini harus di-*escape* kalau kita kebetulan memang ingin secara khusus menangkap karakter itu. Detail di bawah.)
- `.` (sebuah titik) — akan cocok dengan satu karakter apapun kecual garis baru `\n`
- `^`= awal, `$`= akhir — cocok dengan awal dan akhir suatu string.
- `*` — akan mencocokkan 0 atau lebih kemunculan pola di sebelah kirinya. Misal, `ab*` akan menangkap `'a'`, `'ab'`, atau `'a'` diikuti huruf `'b'` sebanyak apapun.
- `+` — akan mencocokkan 1 atau lebih kemunculan pola di sebelah kirinya. Misal, `ab+` akan cocok dengan `'ab'`, atau `'abb'` (atau sebanyak apapun `b`-nya), tapi TIDAK cocok dengan `'a'` saja.
- `?` — akan mencocokkan 0 atau 1 kemunculan pola di sebelah kirinya. Misal, `ab?` akan cocok dengan `'a'` atau `'ab'` saja.
- `*?`, `+?`, `??` — Ini adalah versi "tidak-rakus" dari `*`, `+`, `?`. Tidak-rakus maksudnya dia akan *sesedikit mungkin* meraup ke kanan.
- {`m`} — menentukan bahwa ada *m* perulangan dari pola sebelah kirinya yang cocok. Misal `a{6}` akan cocok dengan enam karakter `'a'`, tapi tidak lima.
- {`m,n`} — mencocokkan sebanyak *m* sampai *n* pengulangan pola di sebelah kirinya. Sebagai contoh, `a{3,5}` akan cocok dengan 3 sampai 5 buah karakter `'a'`. Jika *m* tidak ditulis, maka batas bawahnya dianggap nol, dan jika *n* tidak ditulis, batas atasnya dianggap tak terhingga. Sebagai contoh, pola `a{4,}b` akan cocok dengan `aaaab` atau seribu karakter `'a'` diikuti sebuah `b`. Tapi `aaab` tidak akan cocok (perhatikan bahwa `'a'`-nya cuma 3).
- `[ ]` — kurung siku digunakan untuk mengindikasikan himpunan karekter. Pada sebuah himpunan karakter:
  - Karakter-karakter dapat ditulis satu-persatu, misal pola `[ums]` akan cocok dengan `'u'`, `'m'`, atau `'s'`.
  - Jangkauan/range karakter diindikasikan dengan menulis dua karakter dan dipisahkan dengan `'-'`, misal pola `[a-z]` akan cocok dengan semua huruf kecil ASCII, pola `[0-5][0-9]` akan cocok dengan semua bilangan dua-digit dari `00` sampai `59`, dan

pola `[0-9A-Fa-f]` akan cocok dengan angka hexadecimal manapun. Jika `'-'` di-escape (misal `[a\-z]`) atau ditaruh sebagai karakter pertama atau terakhir (misal `[a-]`), dia akan cocok dengan karakter literal `'-'`.

  - Karkater khusus *kehilangan makna khususnya* di dalam himpunan. Sebagai contoh, pola `[(+*)]` akan cocok dengan yang manapun saja dari karakter literal `'('`, `'+'`, `'*'`, atau `')'`.

  - Kelas karakter seperti `\w` atau `\s` (dijelaskan di bawah) diterima di dalam himpunan.

  - Karakter yang di luar jangkauan dapat dibuat cocok dengan membuat komplemen *"complement"* himpunan itu. Jika karakter pertama di sebuah himpunan adalah `'^'`, maka semua karakter yang *tidak* di himpunan itu akan cocok. Sebagai contoh, pola `[^5]` akan cocok dengan semua karakter kecuali `'5'`. Karakter `'^'` tidak memiliki makna khusus jika tidak ditaruh di posisi pertama sebuah himpunan.

- `|` — Pola `A|B`, di mana `A` dan `B` adalah pola regex apapun, akan membuat regex baru yang mencocokkan pola `A` atau `B`. Jumlah regex yang bisa di-atau-kan ini bisa banyak. Ini bisa dilakukan juga di dalam group (lihat di bawah). Saat string target dipindai ("di-scan"), regex-regex yang dipisahkan oleh `'|'` akan dites dari kiri ke kanan. Ketika satu pola cocok, maka cabang itu diterima. Ini berarti, jika `A` cocok, maka `B` tidak akan diperiksa lagi, meski sebenarnya bisa menghasilkan *match* yang lebih panjang.

- `\w` — huruf w kecil akan cocok dengan karakter kata biasa ("word"): sebuah huruf atau angka atau garis bawah `[a-zA-Z0-9_]`. Perhatikan bahwa meskipun kita boleh menghapalkannya dengan kata "word", pola `\w` hanya akan cocok dengan *satu buah* karakter, tidak seluruh kata. Dan ingat juga bahwa `\W` (dengan huruf W besar) akan cocok dengan karakter non-kata (misal tanda bintang `*`, tanda ampersand `&`).

- `\b` — batas antara kata dan non-kata.

- `\s` — huruf s kecil akan cocok dengan satu karakter putih – spasi, baris baru, 'Enter', tab, dan bentuk `[\n\r\t\f]`. Pola `\S` (huruf S besar) akan cocok dengan karakter non putih.

- `\t, \n, \r` — tab, baris baru, dan 'Enter'

- `\d` — angka `[0-9]`

- `\` — mencegah "kekhususan" suatu karakter; inilah yang dimaksud dengan meng-escape. Misalnya, gunakan `\.` untuk mencocokkan sebuah karakter titik. Gunakan `\\` untuk mencocokkan karakter backslash.

Untuk referensi yang lebih lengkap, silakan kunjungi `docs.python.org/3/library/re.html`

Latihan 7.1 Jalankan kode di atas (halaman 65), pastikan hasilnya adalah seperti yang kamu duga. Di sana, kita memberi perintah seperti ini: "di dalam `string` temukan teks yang memuat `'kata:'` yang diikuti tiga huruf". Perhatikan bahwa ini dicapai dengan menyetel pola = `r'kata:\w\w\w'`. Sekarang jawablah yang berikut:

- Apa yang terjadi kalau kodenya dirubah di baris pertamanya? Seperti ini:
  `string = 'sebuah contoh kata:batagor!!'`
- Bagaimana kalau
  `string = 'sebuah contoh kata:es teh!!'`

□

7.2 Contoh-contoh Dasar

Aturan dasar pencarian pola oleh regex di dalam suatu string adalah:

- Pencarian berjalan di suatu string dari awal sampai akhir.
- Semua pola harus cocok, tapi tidak seluruh string perlu cocok dengan polanya.
- Jika `cocok = re.findall(pola, sebuahString)` berhasil, maka `cocok` berisi daftar teks yang cocok.

```
## Dua baris ini mencari pola 'eee' di string 'teeeh'.
##   Seluruh pola harus cocok, tapi itu bisa muncul di mana saja.
##   Jika berhasil, \texttt{cocok} adalah daftar semua teks yang cocok.
cocok = re.findall(r'eee', 'teeeh') #=>  cocok == ['eee']
cocok = re.findall(r'ehs', 'teeeh') #=>  cocok == []

## . = semua karakter kecuali \n
cocok = re.findall(r'..h', 'teeeh') #=>  cocok == ['eeh']

## \d = karakter angka, \w = karakter huruf atau angka
cocok = re.findall(r'\d\d\d', 't123h di 2019 bulan 02') #=> cocok == ['123', '201']
cocok = re.findall(r'\w\w\w', '@@a*bc#def*tghh!!') #=> cocok == ['def', 'tgh']
```

Latihan 7.2 Pada kode di atas tambahkan baris-baris untuk mengeluarkan hasilnya[2]. Sesudah itu jalankan programnya dan pastikan hasilnya seperti yang kamu duga sebelumnya. Pastikan juga semua pertanyaan "kok bisa seperti itu ya" yang muncul di pikiranmu terjawab tuntas. Tanyalah kepada dosen kuliah atau dosen praktikum jika kamu belum paham akan suatu konsep di latihan ini. □

7.3 Pengulangan dan Kurung Siku

Memprogram regex menjadi lebih menarik saat kita memakai `+` dan `*` untuk membentuk pengulangan (*repetition*) di polanya.

- `+` — 1 atau lebih kemunculan pola di sebelah kirinya. Contoh `'e+'` berarti "satu atau lebih `e`"
- `*` — 0 atau lebih kemunculan pola di sebelah kirinya.
- `?` — cocok dengan 0 atau 1 kemunculan pola di sebelah kirinya.

Ingat, ingat, ada tiga pola kemunculan: 1-atau-lebih, 0-atau-lebih, 0-atau-1.

[2] Misal `print(cocok.group())`

Paling kiri dan paling besar

Pertama-tama pencarian akan menemukan kecocokan yang paling kiri untuk polanya, dan, yang kedua, pencarian ini akan mencoba meraup sebanyak-banyaknya string. Dengan kata lain, `+` dan `*` ini akan meraup sejauh-jauhnya (`+` dan `*` ini sering disebut "rakus"). Jika diinginkan agar regex-nya "tidak rakus", gunakan `+?` dan `*?`.

7.3.1 Contoh pengulangan

```
1  ## e+ = satu atau lebih e, sebanyak-banyaknya.
2  cocok = re.findall(r'te+', 'ghdteeeh') #=>  cocok == ['teee']
3
4  ## Menemukan solusi yang paling kiri, dan dari situ mendorong si tanda '+'
5  ##     sejauh-jauhnya (ingat 'paling kiri dan paling besar'),
6  ## Pada contoh ini, perhatikan bahwa pencarian menemukan dua pola yang tepat.
7  cocok = re.findall(r'e+', 'teeheeee') #=> cocok == ['ee', 'eeee']
8
9  ## \s* = nol atau lebih karakter putih (spasi, tab, dsb.)
10 ## Di sini mencari 3 angka, kemungkinan dipisahkan oleh spasi/tab.
11 polanya = r'\d\s*\d\s*\d'
12 cocok = re.findall(polanya, 'xx1 2   3xx') #=>  cocok == ['1 2   3']
13 cocok = re.findall(polanya, 'xx12  3xx')   #=>  cocok == ['12  3']
14 cocok = re.findall(polanya, 'xx123xx')     #=>  cocok == ['123']
15
16 ## ^ -> cocok dengan awal string, jadi ini tidak akan menemukan:
17 cocok = re.findall(r'^k\w+', 'mejakursi') #=> tidak ketemu, cocok == []
18 ## tapi tanpa ^ dia berhasil:
19 cocok = re.findall(r'k[\w\s]+', 'mejakursi tamu saya')#=> cocok == ['kursi tamu saya']
```

Latihan 7.3 Pada kode di atas tambahkan baris-baris untuk mengeluarkan hasilnya. Sesudah itu jalankan programnya dan pastikan hasilnya seperti contoh. Berikutnya, buatlah analisis cara kerja regex-nya pada baris 7, 12, dan 19. □

Sekarang, bisakah kita meng-ekstrak alamat email dari pengetahuan di atas? Mari kita coba dengan memakai `'\w+'` yang barusan kita pelajari. (Silakan dijalankan!)

```
s = 'Alamatku adalah dita-b@google.com mas'
cocok = re.findall(r'\w+@\w+', s)
print(cocok[0])      ## =>   'b@google'
```

Ternyata ada yang kelewatan! Simbol '`-`' dan '`.`' tidak ditangkap oleh polanya. Di sini kita memerlukan kurung siku.

7.3.2 Kurung Siku

Seperti dijelaskan di atas, kurung siku (karakter '`[`' dan '`]`') dapat dipakai untuk mengindikasikan *sekelompok* karakter, jadi `[abc]` akan cocok dengan `'a'`, `'b'`, atau `'c'`. Kode seperti `\w`, `\s` dan sebagainya tetap berlaku di dalam kurung siku dengan pengecualian simbol titik (`.`). Dia bermakna titik biasa, begitu saja. Titik. Untuk problem ekstraksi alamat email kita di atas, kurung siku merupakan cara mudah untuk menambahkan `'.'` dan `'-'` pada himpunan karakter yang muncul di sekitaran simbol @ dengan pola `[\w.-]+@[\w.-]+` untuk mendapatkan alamat email yang utuh:

```
cocok = re.findall(r'[\w.-]+@[\w.-]+', s)
print(cocok[0])     ## => 'dita-b@google.com'
```

Latihan 7.4 Pola di atas sudah bisa mengekstrak email dengan cukup bagus. Namun masih ada yang kurang. Nah, pola berikut bisa mengekstrak dengan lebih baik (meski tetap saja belum sempurna). Buatlah analisisnya!

```
pola = r'[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+'
```

Peningkatan apa yang diberikan oleh pola baru ini dibandingkan dengan pola sebelumnya? □

7.4 Ekstraksi secara Group

Fasilitas *grouping* di regex memungkinkan kita untuk memilah-milah bagian teks yang telah cocok. Misal pada problem alamat email di atas kita ingin mengekstrak *username* dan *host*-nya secara terpisah. Untuk melakukan hal ini, perlu kita tambahkan tanda kurung `()` di seputar username dan host di polanya, seperti ini: `r'([\w.-]+)@([\w.-]+)'`.

Di sini, tandakurungnya tidak mengubah pola yang akan cocok, tapi dia akan mengelompokkan teks-teks yang cocok. Jika pencarian berhasil (lihat contoh di bawah), object `cocok` akan berisi tuple-tuple yang berisi semua temuan. Tiap-tiap tuple akan berisi pasangan sesuai pola `grouping`-nya.

Di contoh berikut ini, perhatikan bahwa `cocok[0]` adalah sebuah tuple. Lalu `cocok[0][0]` adalah teks cocok yang terkait pasangan tanda kurung pertama, dan `cocok[0][1]` adalah teks cocok yang terkait pasangan tanda kurung kedua. Jadi, setiap tuple mempunyai panjang 2 dan berisi username dan host, misal `('sri','google.com')`.

```
s = 'Alamatku adalah dita-b@google.com mas'
cocok = re.findall(r'([\w.-]+)@([\w.-]+)', s) ## perhatikan posisi () di polanya
cocok  ##  adalah [('dita-b', 'google.com')]
## Bisa kita pilah satu per satu seperti ini:
cocok[0][0]  ## 'dita-b'
cocok[0][1]  ## 'google.com'
```

Alur kerja yang biasa dilakukan adalah pertama-tama kita tulis regex untuk teks yang kita cari, lalu menambahkan kelompok tandakurung untuk mengekstrak bagian-bagian yang kita inginkan.

Sekarang cobalah yang berikut.

```
## Kita punya banyak alamat email
s = 'Alamatku sri@google.com serta joko@abc.com ok bro. atau don@email.com'

## Di sini re.findall() mengembalikan sebuah list beranggotakan string alamat
pola = r'[\w\.-]+@[\w\.-]+'
e = re.findall(pola, s)
print(e)
##=> e akan berisi ['sri@google.com', 'joko@abc.com', 'don@email.com']
```

Sekarang kita siap untuk meng-group polanya menjadi username dan host.

```
pola = r'([\w\.-]+)@([\w\.-]+)'
e = re.findall(pola, s)
print(e)     ##==> sekarang bagaimanakah hasilnya?
## Atau kita cetak satu per satu:
for tup in e:
    print('user', tup[0], 'dengan host:', tup[1])
```

Latihan 7.5 Tambahkan beberapa kata dan alamat email pada string `s` di atas. Jalankan, dan pastikan alamat email yang baru kamu sertakan akan tertangkap dan 'terbelah' oleh regex-nya. Buatlah analisisnya. □

7.4.1 Pencarian dalam berkas

Untuk berkas, mungkin kamu punya kebiasaan menulis sebuah loop yang memutari baris demi baris, lalu memanggil `findall()` pada tiap baris. TAPI, ada yang lebih baik! Berikan semuanya pada `findall()` dan dia akan mengembalikan *semua* teks yang cocok. Berikut ini adalah ilustrasinya.

```
f = open('test.txt', 'r', encoding='latin1') ## membuka file.
teks = f.read()
f.close()
p = r'sebuah pola'        ## ini polanya.
## memberikan seluruhnya ke findall()
## dia mengembalikan list beranggotakan string yang cocok
strings = re.findall(p, teks)
```

Beberapa latihan didedikasikan untuk ini. Silakan dikerjakan.

7.5 Referensi lebih lanjut

Regex adalah topik yang luas. Untuk belajar lebih lanjut, silakan buka

- docs.python.org/3/library/re.html
- docs.python.org/3/howto/regex.html
- developers.google.com/edu/python/regular-expressions
- www.tutorialspoint.com/python/python_reg_expressions.htm

7.6 Soal-soal untuk Mahasiswa

Sebelum mengerjakan soal-soal di bawah, kerjakan dulu latihan-latihan di atas.

1. **Meng-ekstrak kata-kata dengan awalan 'me'**. Bukalah halaman web id.wikipedia.org/wiki/Indonesia. Salin seluruh tubuh artikel itu dan tuang ke Notepad[3]. Simpan sebagai `Indonesia.txt`. Tugasmu adalah mengekstrak semua kata yang mempunyai awalan

[3]Memakai `Ctrl-A`, lalu `Ctrl-C`, lalu `Ctrl-V` juga boleh.

'me', lalu menyimpannya dalam sebuah tuple. Perhatikan bahwa 'me' bisa muncul di awal, tengah ataupun akhir kalimat[4].

2. **Meng-ekstrak kata-kata dengan awalan 'di'**. Sama seperti di atas, tapi yang kamu ekstrak adalah kata-kata dengan awalan 'di'. Perhatikan, yang dimaksud di sini adalah *awalan* 'di' (menunjukkan kata kerja pasif), bukan *kata depan* 'di' (menunjukkan keterangan tempat), yang kita bahas di nomer di bawah.

3. **Meng-ekstrak kata depan 'di' dan tempat yang ditunjuknya**. Di sini tugasmu adalah mengekstrak – masih dari berkas `Indonesia.txt` – kata depan 'di' *beserta satu kata yang mengikutinya*. Misal, dari string

   ```
   'Saya dilahirkan dan dibesarkan di sini, di Indonesia, sebuah negeri indah
   permai di mana para saudagar di masa lampau berdatangan.'
   ```

   akan ter-ekstrak `('di sini', 'di Indonesia', 'di mana', 'di masa')`

4. **Meng-ekstrak data dari berkas `.html`**. Tugasmu di sini adalah membuat sebuah *list of tuples* yang memuat nama-nama negara beserta *Innovation index*-nya. Kunjungilah halaman `en.wikipedia.org/wiki/Knowledge_Economic_Index`. Pastikan halaman itu memuat sebuah tabel besar hasil penelitian. Kerjakan langkah-langkah berikut.

   - Simpanlah halaman itu sebagai berkas html murni, dengan nama `KEI.html`[5].
   - Bukalah berkas `KEI.html` itu dengan Notepad++ atau Notepad. Perhatikan segmen-segmen di file itu yang mempunyai isi seperti di bawah

     ```
     ...
      </span><a href="/wiki/Belgium" title="Belgium">Belgium</a></td>
     <td>8.73</td>
     <td>8.70</td>
     <td>8.82</td>
     <td>8.96</td>
     <td>9.14</td>
     <td>8.02</td>
     <td>16</td>
     </tr>
     <tr>
     ...
     ```

   - Akses file itu menggunakan `f=open('KEI.html', 'r', encoding='latin1')` dan baca file menggunakan `teks = f.read()`. Lalu tutup aksesnya menggunakan `f.close()`.
   - Sebagai awalan, ekstraklah *semua* nama-nama negara yang ada di berkas itu memakai regex, dengan *memperhatikan pola di sekitaran nama-nama negara*[6].

[4] Perhatikan pula bahwa regex-mu juga akan menangkap kata-kata seperti 'meskipun', 'mereka', 'mesin', 'Meulaboh', dan 'Merauke'. Mereka ini tentu saja bukan kata kerja. (Nomer di bawahnya juga mempunyai masalah serupa.) Untuk sekarang, tidak mengapa kalau kata-kata itu tertangkap. Bisakah kamu memikirkan suatu cara agar kata-kata itu diabaikan oleh regex-mu?

[5] Caranya bisa dengan meng-klik kanan di halaman itu lalu klik "Save as..." (untuk Google Chrome), atau "Save Page As..." (untuk Mozilla Firefox). Ketika muncul sebuah *dialog box*, simpan di folder kerjamu sebagai KEI.html dan – di bawahnya – pilih "Web page, HTML Only".

[6] Lihatlah bahwa kamu mempunyai tiga pilihan untuk meng-ekstrak nama negara itu:

- Sekarang ekstraklah kolom "Innovation" untuk semua negara di tabel itu (untuk contoh Belgium di atas, kolom ini adalah kolom yang angkanya 8.96) memakai regex[7]. Sebagai tambahan, ubahlah string `'8.96'` menjadi float.
- Dengan konsep group (memakai tanda kurung `(...)` di tempat yang tepat, lihat Section 7.4 di halaman 70), ekstraklah nama negara beserta *Innovation Index*-nya, lalu buatlah kode seperlunya untuk memodifikasinya, sehingga akhirnya kamu mempunyai *list of tuples* seperti berikut

```
[('Belgium', 8.96), ('Malaysia', 6.83), ('Indonesia', 3.32), ... ]
```

Penting: di bab ini kita membahas telah regular expression untuk mem-*parse* teks dengan pola tertentu. Di bagian latihan, kita mencoba mem-`parse` file html untuk diekstrak informasinya. Namun sebenarnya *regular expression* tidaklah handal untuk mem-*parse* file html.

Terdapat modul python khusus untuk *parsing* suatu file html, namanya *Beautiful Soup*. Jika kamu akan melakukan *html parsing* ataupun *web scraping*, gunakan *Beautiful Soup* ini.

– apakah memakai pola `/wiki/Negara`,
– atau pola `title="Negara"`,
– atau pola `">Negara</a></td>`.

Salah satunya "lebih benar" dari yang lainnya.

[7]Kira-kira bagaimanakah strateginya? Petunjuk kecil: perhatikan bahwa angka 8.96 di atas tercantum empat `<td>` sesudah `</a></td>`. Dan, bagaimanakah kamu membuat pola "satu angka diikuti titik diikuti dua angka"?