12. Най-хубавото

14 ноември 2022

Традиционно - въпрос

```
class Харем:

    управителка = "Валиде султан"

    def __init__(self):
        self.управителка = "Махидевран султан"

харем = Харем()

del харем.управителка
print(харем.управителка)

del харем.управителка
print(харем.управителка)

# ?
```

```
'Валиде султан'
```

```
Traceback (most recent call last):
  File "<pyshell#116-еп.>", line 1, in <module>
    del харем.управителка
АтрибутХатасъ: йок харем.управителка
```

Регулярни изрази

Какви знания ще ни трябват за днес:

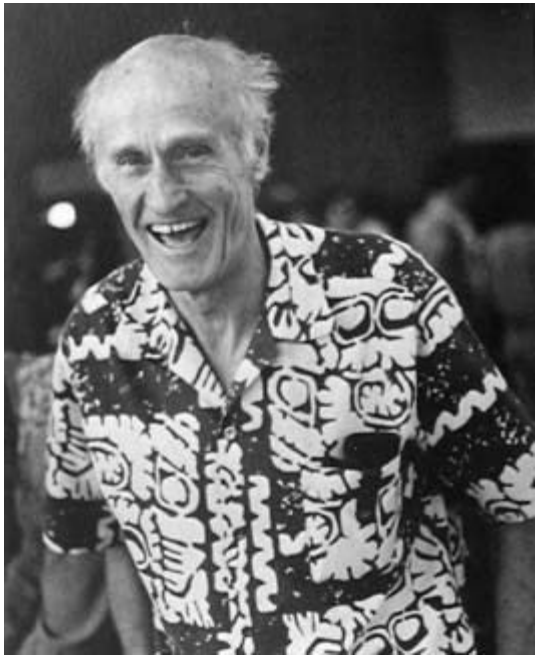
- Регулярни езици
- Регулярни граматики
- Йерархия на Чомски
- Недетерминирани крайни автомати
- *Всъщност това последното е лесно, ето:*
  $\mathcal{N} = \langle \Sigma, Q, Q_{start}, \Delta, F \rangle$

След като ви развалихме съня тази вечер

- Нищо от това на предния слайд няма да ни трябва.
- Това ще го учите в “Езици, автомати и изчислимост”...
- Всъщност, ще си говорим за:
  - Очевидно, регулярни изрази
  - С какво ще ни улеснят живота - в и извън рамките на Python
  - С какво ще ни вгорчат живота
  - Особенности / синтаксис в Python
  - Плюс една дребна задачка (вече традиционна)

Малко обща култура

- Концепцията се ражда през 50-те години на миналият век.
- By Стивън Коул Клийни...
- This guy —>
- Вижте го какъв е добряк...
- Как може да са нещо лошо?
- Навлизат в ИТ света през Unix / POSIX / Perl



Употреба

- Search engines
- Search and replace в текстовите редактори
- Туулове за обработка на текст като sed и AWK
- Лексикален анализ
- Защото са яки
- To save the day

Проблематика

- Търсене на по-сложна последователност от символи в низ
- Заместване на такива последователности с нещо друго
- Проверка дали даден низ отговаря на определени условия
- Защото са най-якото нещо евър

Примерен проблем

- Искаме да проверим дали даден низ съдържа валиден телефонен номер.
- Това означава:
  - Трябва да съдържа само цифри
  - Може да започва с код на населеното място: 02, 042 или 09172
  - След кода, дължината му може да е между 5 и 7 цифри
  - Самият номер (след кода) не може да започва с 0, 1, 2, 3 или 4

Вариант 1

```
def validate_phone_str(number):
    if number[:2] == '02':
        return validate_phone_str(number[2:])
    elif number[:3] == '042':
        return validate_phone_str(number[3:])
    elif number[:5] == '09172':
        return validate_phone_str(number[5:])
    if all([c.isdigit() for c in number]) and number[0] > 4:
        return 5 <= len(number) <= 7
    return False
```

Вариант 2

```
def validate_phone_re(number):
    pattern = r'^(02|042|09172)?[5-9]\d{4,6}$'
    return bool(re.search(pattern, number))
```

- Cooler, eh?

Преди това, за тези, които знаят как работят регулярните изрази, предизвикателство:

```
r'^(11+)(\1)+$'
```

До края на настоящият час отгатнете какво прави този регулярен израз. :)

Терминология

- **Основно:** "шаблон" (pattern), още "регулярен израз"
- Специални (meta) символи
- Екраниране (escape-ване) на специалните символи
- Квантори (quantifiers) и повторения
- Класове от символи
- Групи
- Флагове (modifiers) на шаблона

В Python

- `import re` — модулът, реализиращ PCRE-функционалността.
- Escape-ване на специални символи: чрез \
- За задаване на шаблоните обикновено се ползват raw низовете.
- Сиреч сурови стрингове - няма нужда да ескейпваме наклонените черти.
- Пример: `r'\s+'`

Шаблони

- В шаблона всеки символ, освен някои специални, означава себе си.
- Then again, цялата магия е в специалните символи:
  `. \ | ( ) [ ] { } + \ ^ $ * ?`
- `\` пред специален символ го прави неспециален такъв.
- Някои символи са специални само в определен контекст (напр. `-`).

Шаблони

- Примерите ще демонстрираме чрез наша функция `matcher()`.
- Не е част от стандартната библиотека на Python, ползваме я за удобство.
- Ще ви покажем 4-те ѝ реда код по-късно.
- Сигнатура: `matcher(pattern, string)`.

Mage guild level 1 - повторения

Важат за непосредствено предхождащия ги символ/клас/група. Нека го означим с `s`.

- `s*` означава нула или повече повторения на `s`.
- `s+` търси едно или повече повторения на `s`.
- `s?` съвпада с нула или едно повторение на `s`.
- `s{m,n}` означава между `m` и `n` повторения на `s`, където можем да пропуснем `m` или `n`.
- `s{,n}` има смисъл на нула до `n` повторения, а `s{m,}` — поне `m` повторения.

Примери

```
matcher('o+', 'Gooooooooogle')           # 'G(oooooooo)gle'
matcher('[hH]o+', 'Hohohoho...')         # '(Ho)hohoho...'
# Хм. Не искахме точно това. По-скоро:
matcher('([hH]o)+', 'Hohohoho...')       # '(Hohohoho)...'
matcher('([hH]o){2,3}', 'Hohohoho...')   # '(Hohoho)ho...'
```

Non-greedy?

По подразбиране - алчно търсене за съвпадение (greedy). Деактивира се с ? след квантора.

```
matcher('[hH]o+', 'Hooooooohohooo...')    # '(Hooooooo)hohooo...'
matcher('[hH]o+?', 'Hooooooohohooo...')   # '(Ho)ooooohohooo...'
```

Скоби

Скобите (и) се използват за логическо групиране на части от шаблона с цел:

- Контролиране областта на влияние на дадена операция
- Възможност за референция към "оградето" в скобите
- Задаване на по-специални (и не толкова често употребявани) конструкции

Повече за групите - след малко.

Mage guild level 2 - специални символи

- `.` съвпада с един произволен символ. По подразбиране символите за нов ред не се включват в тази група.
- `^` съвпада с началото на низ (или на ред, ако се работи в `MULTILINE` режим).
- `$` съвпада с края на низ (или на ред, ако се работи в `MULTILINE` режим).

Колец (pipe)

- `|` има смисъл на или, например:

```
matcher('day|nice', 'A nice dance-day.')   # 'A (nice) dance-day.'
matcher('da(y|n)ce', 'A nice dance-day.')  # 'A nice (dance)-day.'
```

NB! Единствено `|` се прилага не над непосредствените му символи/класове, а на целия низ отляво/отдясно:

```
matcher('ab|c|e', 'abcdef')     # '(ab)cdef'
matcher('am|c|e', 'abcdef')     # 'ab(c)def'
matcher('a(m)|c|e', 'abcdef')   # 'ab(c)def'
```

Дребна забележка

- Стандартно ни се връща първият намерен match.

```
matcher('da(y|nce)', 'A nice dance-day.')   # 'A nice (dance)-day.'
```

Mage guild level 3 - символни класове

- Набор от символи, заграден от `[` и `]`, например `[aeoui]`.
- Съвпадат с точно един от символите, описани в класа, например:

```
matcher('[aeoui]', 'Google')                # 'G(o)ogle'
```

- Отрицание на клас — посредством `^` в началото на класа:

```
matcher('[^CBL][aeoui]', 'Cobol')           # 'Co(bo)l'
```

- Диапазони от символи:

```
matcher('[0-9]{1,3}-[a-z]', 'Figure 42-b') # 'Figure (42-b)'
matcher('[^a-zA-Z-]', 'Figure-42-b')        # 'Figure-(4)2-b'
```

Предефинирани класове

- `\d` — една цифра; същото като `[0-9]`.
- `\D` — един символ, който не е цифра; същото като `[^0-9]`.
- `\s` — един whitespace символ — `[\t\r\n\f\v]`.
- `\S` — един символ, който не е whitespace — `[^\t\r\n\f\v]`.
- `\w` — една буква или цифра.
- `\W` — един символ, който не е буква или цифра.
- `\b` — нула символа, но граница на дума.
- И други.

Няколко примера

```
matcher(r'\d+', 'Phone number: 5551234')
# 'Phone number: (5551234)'

matcher(r'\w+', 'Phone number: 5551234')
# '(Phone) number: 5551234'

matcher(r'\s+', 'Phone number: 5551234')
# 'Phone( )number: 5551234'
```

Armageddon level shit - пак групи

- Казахме вече - групите са частите от даден шаблон, оградени в `(` и `)`.
- Към тях можем да се обръщаме и от самия шаблон чрез специалните класове `\1` — първата група, `\2` — втората и така нататък.

Няколко примера:

```
matcher(r'(\w+).*\1', 'Matches str if str repeats one of its words.')
# 'M(atches str if str repeat)s one of its words.'
```

- Хм. Не точно. Нека опитаме пак:

```
matcher(r'(\b\w+\b).*\1', 'Matches str if str repeats one of its words.')
# 'Matches (str if str) repeats one of its words.'
```

Методи на re

- `re.search()` — проверява дали даден низ съдържа текст, отговарящ на зададения шаблон, връща `re.Match` обект.
- `re.match()` — същото както горното, само че се търси за съвпадение в началото на низа, връща `re.Match` обект.
- `re.findall()` — връща като списък всички съвпадения на шаблона в дадения низ.
- `re.finditer()` — същото като горното, но връща итератор.

Групите в контекста на re.Match

- `re.Match` обектите съдържат подробна информация за групите, които сме мачнали:

```
re.search(r'(. )( .)(\1\2)*', 'a  ba  ba  bc')
# <re.Match object; span=(0, 12), match='a  ba  ba  b'>

re.search(r'(. )( .)(\1\2)*', 'a  ba  ba  bc').group()
# 'a  ba  ba  b'

re.search(r'(. )( .)(\1\2)*', 'a  ba  ba  bc').groups()
# ('a ', ' b', 'a  b')

re.search(r'(. )( .)(\1\2)*', 'a  ba  ba  bc').group(1)
# 'a '
```

Още за re.Match обектите

- `group()` — връща частта от низа, отговаряща на шаблона (и още...).
- `start()` — връща началото на съвпадението в низа.
- `end()` — връща края на съвпадението в низа.
- `span()` — връща `(start, end)` под формата на tuple.

Вече можем да го покажем - кодът на matcher()

```
def matcher(regex, string):
    match = re.search(regex, string)
    if match is None:
        return string
    start, end = match.span()
    return f'{string[:start]}({string[start:end]}){string[end:]}'
```

Highest-level magic (e.g. Implosion)

С други думи - групи за напреднали

- `(?:...)` — използване на скоби, без да се създава група.
- `(?P<name>...)` — текстът, отговарящ на групата, може да бъде достъпван чрез име, вместо чрез номер.
- `(?P=name)` — търси съвпадение за текста, намерен по-рано от групата, кръстена name.
- `(?#...)` — коментар, игнорира се.
- `(?=...)` — съвпада, ако ... следва, но не го "консумира" (look-ahead).
- `(?!...)` — съвпада, ако ... не следва (negative look-ahead).
- `(?<=...)` — съвпада ако ... стои преди следващата част от шаблона (look-behind).
- `(?<!...)` — съвпада ако ... **не** стои преди следващата част от шаблона (neg. `^`)
- `(?(id/name)yes|no)` — търси за шаблона 'yes', ако групата с номер/име съвпада, или с (опционалния) шаблон 'no' иначе.

(?:...) - групи, без да са групи?!

```
re.search(r'(\w+) \d', 'The 4 Horsemen of the Apocalypse').group()
# 'The 4'
re.search(r'(\w+) \d', 'The 4 Horsemen of the Apocalypse').groups()
# ('The',)
```

```
re.search(r'(?:\w+) \d', 'The 4 Horsemen of the Apocalypse').group()
# 'The 4'
re.search(r'(?:\w+) \d', 'The 4 Horsemen of the Apocalypse').groups()
# ()
```

(?P<name>...) - именувани групи

```
re.search(r'\w+\s*(?P<number_of_horses>\d)\s*\w+', 'The 4 Horsemen of the
Apocalypse').group()
# 'The 4 Horsemen'

re.search(r'\w+\s*(?P<number_of_horses>\d)\s*\w+', 'The 4 Horsemen of the
Apocalypse').groups()
# ('4',)

re.search(r'\w+\s*(?P<number_of_horses>\d)\s*\w+', 'The 4 Horsemen of the
Apocalypse').groupdict()
# {'number_of_horses': '4'}

re.search(r'\w+\s*(?P<number_of_horses>\d)\s*\w+', 'The 4 Horsemen of the
Apocalypse').group('number_of_horses')
# '4'
```

(?=...) - look-ahead

```
re.search(r'[Tt]he', 'The 4 Horsemen of the Apocalypse').group()
# 'The'

re.search(r'[Tt]he(?=\s*Apocalypse)', 'The 4 Horsemen of the
Apocalypse').group()
# 'the'
```

(?!...) - negative look-ahead

```
re.search(r'[Tt]he', 'The 4 Horsemen of the Apocalypse').group()
# 'The'

re.search(r'[Tt]he(?!\s*4)', 'The 4 Horsemen of the Apocalypse').group()
# 'the'
```

(?<=...) - look-behind

```
re.search(r'[Tt]he', 'The 4 Horsemen of the Apocalypse').group()
# 'The'

re.search(r'(?<=\s)[Tt]he', 'The 4 Horsemen of the Apocalypse').group()
# 'the'
```

- Има и negative lookbehind.
- И двете са пипкави...

Методи на re (2)

- `re.sub(pattern, repl, string, count=0)` — заместване в низ, на база на шаблон.
- `re.split(pattern, string, maxsplit=0)` — разделяне на низ на парчета, на база на шаблон.
- `re.escape(pattern)` — escape-ва всички специални за регулярен израз символи.
- Пример: `re.escape('a(a)\s+')` ще върне `'a\\(a\\)\\\\s\\+'`
- Затова най-често ползваме raw strings.
- Още: `help(re)`

И за да знаете че съществуват - флагове

- `re.I` (`re.IGNORECASE`) — case-insensitive търсене.
- `re.L` (`re.LOCALE`) — кара `\w`, `\W`, `\b`, `\B` да зависят от текущия locale.
- `re.M` (`re.MULTILINE`) — кара "`^`" да съвпада както с начало на низ, така и с начало на ред, докато "$" ще съвпада с край на ред или края на низа.
- `re.S` (`re.DOTALL`) — "`.`" ще съвпада с всеки символ, включително и нов ред.
- `re.X` (`re.VERBOSE`) — режим на игнориране на white-space и коментари (за по-дългички RE).
- `re.A` (`re.ASCII`) — кара `\w`, `\W`, `\b`, `\B`, `\d`, `\D` да отговарят на съответните ASCII-класове.

Кога да НЕ използваме регулярни изрази?

- Ако имате по-добра опция.
- Примери - xml, html.
- С други думи - не откривайте топлата вода.
- По-прост пример - ако можете да използвате един `or` и 2 `in` оператора - може би не си заслужава усилието за регулярен израз.

Кога още да НЕ ползваме регулярни изрази?

Mail::RFC822::Address: regexp-based address validation

Mail::RFC822::Address is a Perl module to validate email addresses according to the **RFC 822** grammar. It provides the same functionality as RFC::RFC822::Address, but uses Perl regular expressions rather that the Parse::RecDescent parser. This means that the module is much faster to load as it does not need to compile the grammar on startup.

The grammar described in RFC 822 is surprisingly complex. Implementing validation with regular expressions somewhat pushes the limits of what it is sensible to do with regular expressions, although Perl copes well. The regular expression below shows the complexity, although its inclusion on this page has caused some confusion:

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t]
)+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:
\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(
?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[
\t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\0
31]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\
](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+
(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:
(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z
|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)
?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\
r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
 \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)
?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t]
)*))*:(?:(?:\r\n)?[ \t])?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t]
)+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*
*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+
|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r
\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:
\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t
]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031
]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](
?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?
:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?
:\r\n)?[ \t])*))*\>(?:(?:\r\n)?[ \t])*)|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?
:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?
[ \t]))*"(?:(?:\r\n)?[ \t])*)*:(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\]
\000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|
\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>
@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"
(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t]
)*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?
:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[
\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-
\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(
```

Кога още да НЕ използваме регулярни изрази?

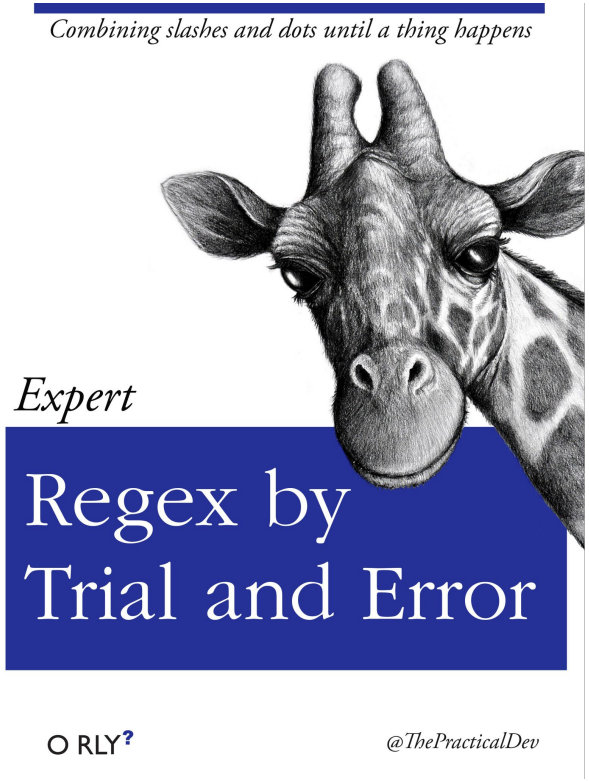
- С други думи - когато сложността на задачата надхвърля някакви разумни граници.
- Ако ви трябва да парсвате C# код - не ви трябват регулярни изрази, а **парсър.**

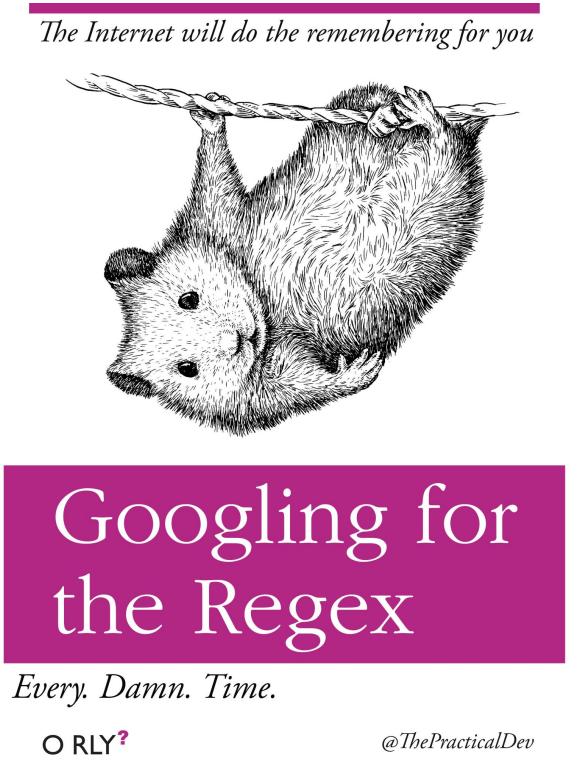
Кога още още да НЕ използваме регулярни изрази?

- Когато очаквате кодът да претърпи много промени и искате да се поддържа лесно.
- Примери от живия живот:

```
r'\s*\{{{{\s*match\s*=>\s*"{0}_{{}}[0x]\.\*"'.format(cm_component)

r'{0}typ.h\s*,\s*VPSRCS\s*=(([\t ]*(\w+)[\t ]*(\\[
\t]*\n)?)+)'.format(interface)

r'^\s*([A-Z]{2,6}xE?VPx[A-Zx]+)typ.h[^\\\n]*(\\\s*\n[^\\\n]*)*\b%s\b.*$'.forma
t(item)

lambda text: '_'.join([word[0].lower() for word in
re.findall(r'([A-Z][a-z0-9]+|[A-Z]+(?=([_A-Z]+?[a-z0-9]?|$))|[^_A-Z]+)',re.sub
('[\s\-\.]', '_', re.sub(r'[^\w\s\-\.]', '', text)))])
```

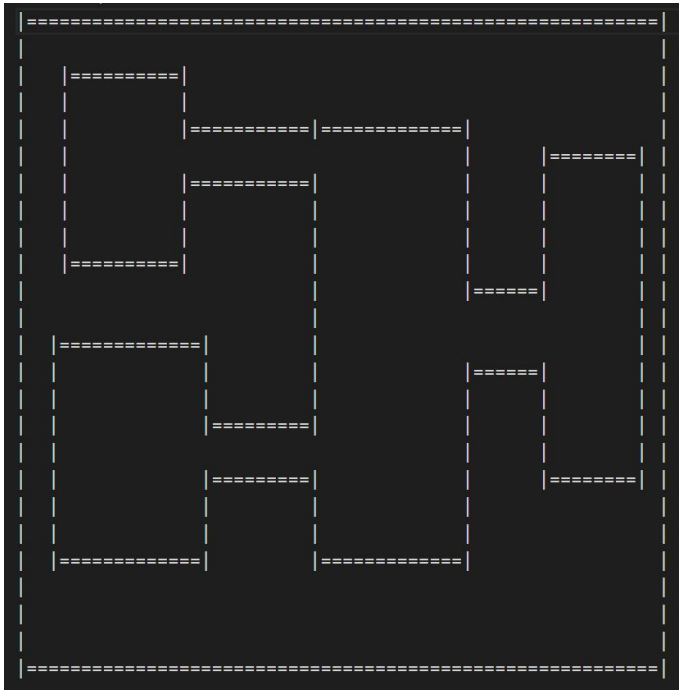
Четиво за регулярни изрази



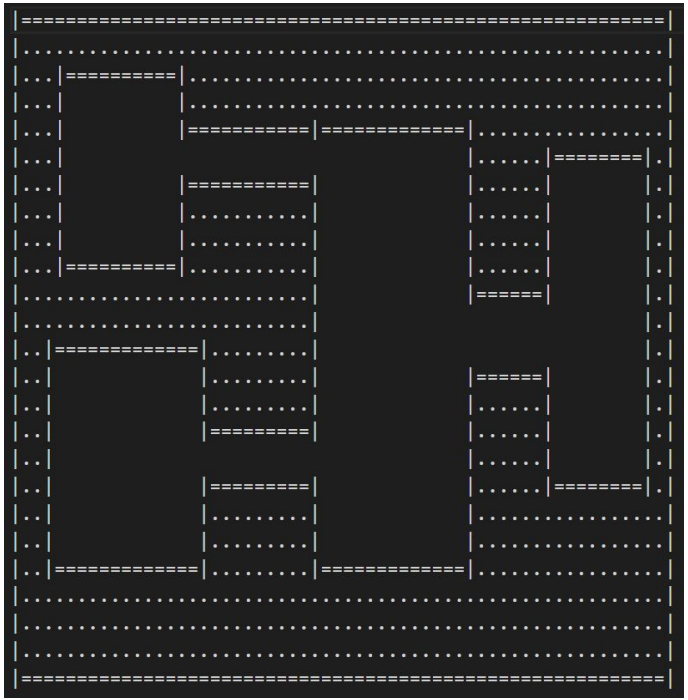


Още едно питане

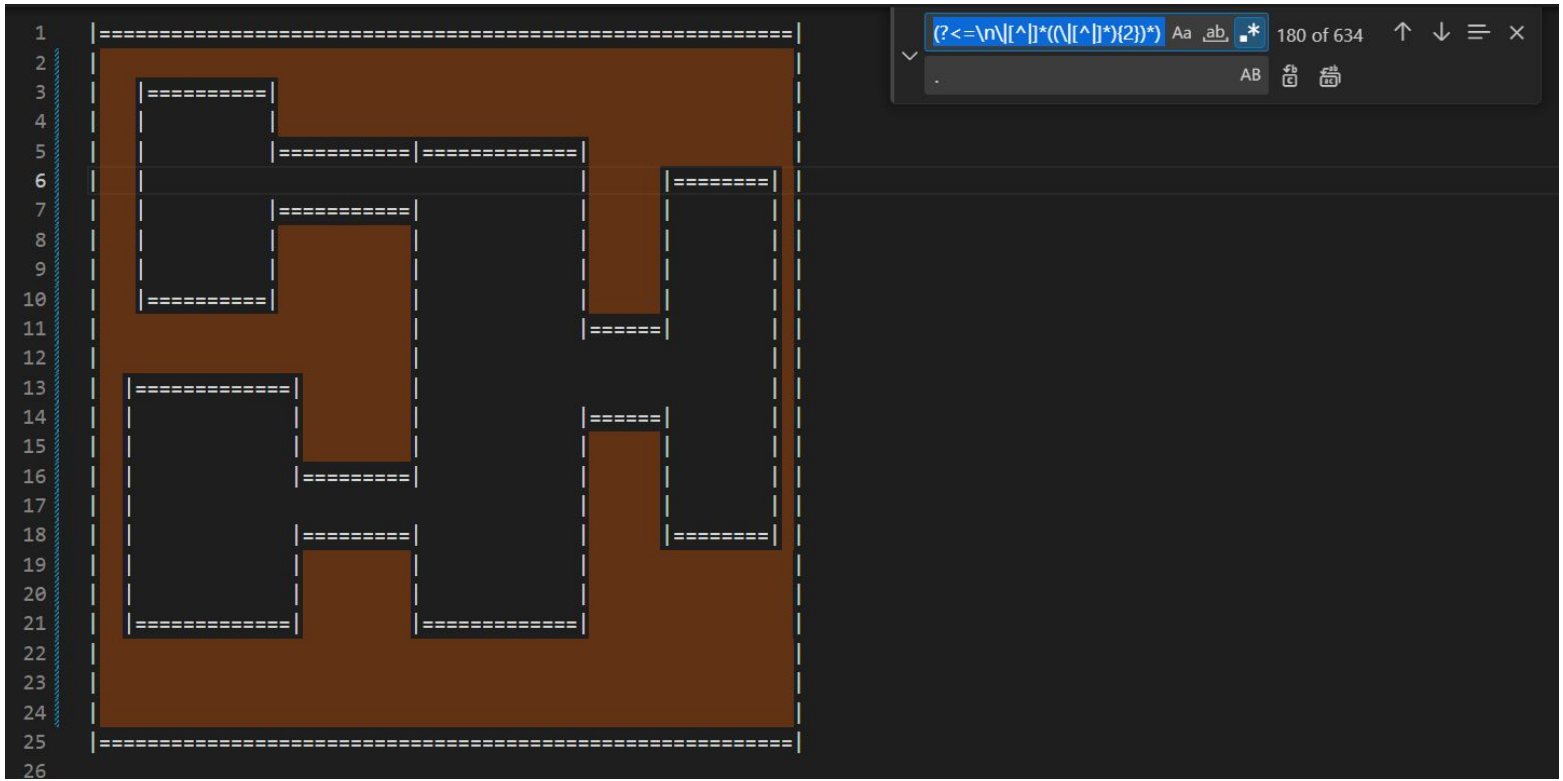
Как да направим от това:



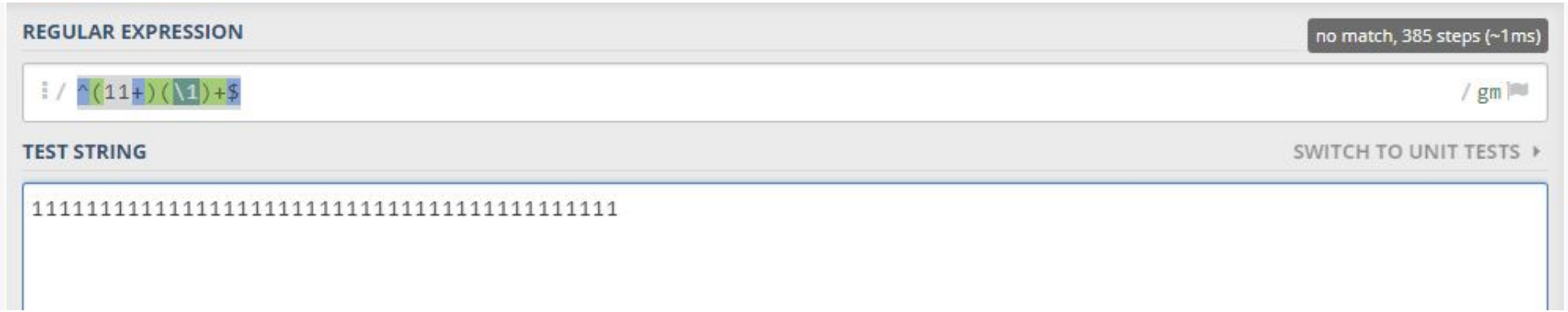
Това?

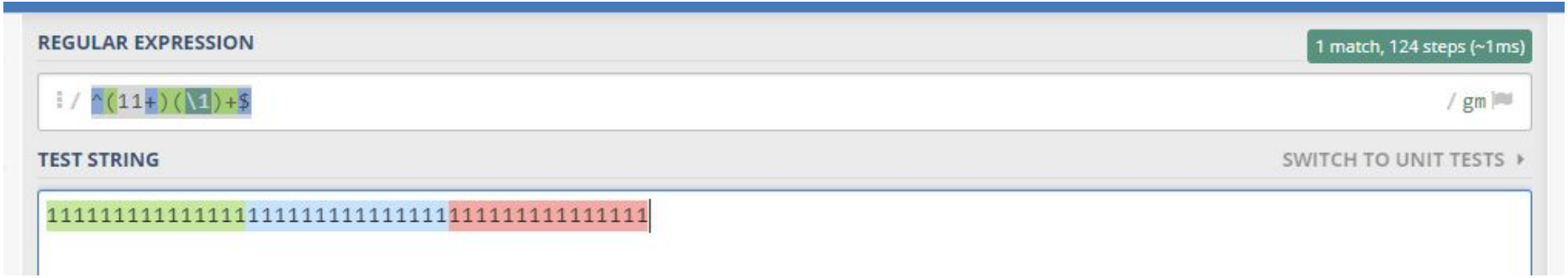


Елементарно - '(?<=\n\|[^|]*((\|[^|]*){2})*) '



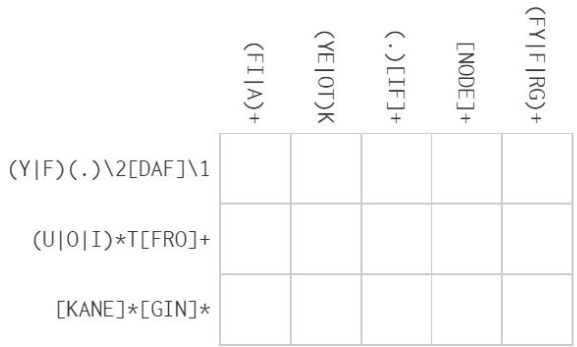
Помните ли задачката в началото?





За маз... хората, които знаят как да се забавляват

https://regexcrossword.com/



Главоблъсканица по контролното

- Разделете следния текст на въпроси
- Вече разделените въпроси разделете на заглавие и отговори
- Вече разделените отговори разделете на текст и указател за вярност

(1) Кога ще бъде контролното?

- Днес
- Контролното мина
- Няма да има контролно. Това да не ти е УНСС?
+ В четвъртък

(2) Какво ще има на контролното?

+ Въпроси за Python
+ Въпроси за Monty Python
- Въпроси за майка ти
- Време за отговор на всички въпроси

Валидатор за успешен полов акт

- На база стенанията на партньора, проверете дали е задоволен(а)
- Важните стенания са “ах”, “ех”, “ох”, “ъх”. Останалото е пълнеж и не ни интересува
- Ако след дадено стенание от списъка последва стенание, което се намира по-рано в списъка, то това се счита за деградация и половият акт е неуспешен
- Ако през целия текст стенатията градират, всичко е било “Уау”
- Стенания с удължена гласна, като “аааах” са валидни репрезентации на кратката си версия

Пример 1: Ах, аах, ааах, шнеле, шнеле, ех, ееех, ох - ъх и те така
Пример 2: Ехоо, ъх, ах
Пример 3: Ех, Митко, Миткооо, Митко
Пример 4: Ох, банана, ах, банана, кеф ми става, щом го хвана
Пример 5: Ах, ах - умирам за тебе и планове правя,
ах, ах - една целувка как да откраднa?
Ох, не е ли лудост това, ох, да бъдем заедно!
Ще бъдеш ли мой?
/Еми Стамболова - Ах-Ах/

Валидатор за Палиндром

- Напишете регулярен израз, който проверява дали дадена дума е палиндром, или не
- Ако не става, използвай само конкретен брой символи
- Ако и това не става - да си ходим?

Въпроси?