```
[#capture_number
] Reasons to Switch to re3, the
[#capture_number
]th Made Me [case_insensitive ['laugh' | 'cry']
]![#capture_number=[capture 1+ #digit]]
```

Aur Saraf

Regex is Awesome!

```
^(\d+) reasons to use regular expressions, the (\d+)(?:st|nd|rd|th) made me cry!$
```

```python
def match_n_reasons(s):
    try:
        i, rest = s.split(' reasons to use regular expressions, the ')
        assert i.isdigit()
        nth, nothing = rest.split(' made me cry!')
        j, th = jth[:2], jth[2:]
        assert all([j.isdigit(), th in 'st nd rd th'.split(), not nothing])
        return i, j
    except:
        return None
```

Invented 1986 by Henry Spencer and Larry Wall

~~Invented 1986 by Henry Spencer and Larry Wall~~

Invented 1968 by Ken Thompson

~~Invented 1986 by Henry Spencer and Larry Wall~~

~~Invented 1968 by Ken Thompson~~

Invented 1956 by Stephen Cole Kleene

Regex Syntax is Horrible!

```
\b(-?\d+)(?:.(\d+))?(?:[Ee](-?\d+))?\b
```

Quick, what does this do?

Where is the bug?

```
\b(-?\d+)(?:\.(\d+))?(?:[Ee](-?\d+))?\b
```

Arcane Symbols

Bugs are hard to spot

Following subjects verbs are 

```
w{3}\.\w{1,3}\.com
```

Quick, what does this do?

Where’s the bug?

```
w{3}\.\w{1,3}\.com
```

www.o_o.com?

www.١٢٣.com?

Data unseparated from Meta

```
\b(25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
 (25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
 (25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
(25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d)\b
```

Quick, what does this do?

```
\b(25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
 (25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
 (25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
(25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d)\b
```

Can’t DRY

Can’t document

No support for common cases

Unmaintainable.

```
urlpatterns = patterns('',
    url(r'^tts/(?P<language>\w*)/(?P<phrase>.*)/$',
        tts.views.tts, name='tts'),
    url(r'^tts/(?P<phrase>.*)/$',
        tts.views.tts, name='tts-en'),
```

A weird language from the 60s to learn

Easy to fall into traps

Full of wat

```
urlpatterns = patterns('',
    url(r'^tts/(?P<language>\w*)/(?P<phrase>[^/]*)/$',
        tts.views.tts, name='tts'),
    url(r'^tts/(?P<phrase>[^/]*)/$',
        tts.views.tts, name='tts-en'),
```

A weird language from the 60s to learn

Easy to fall into traps

Full of wat

Regex Syntax is Horrible!

No surprise, it was created without a design process in 1968

Your mission, if you choose to accept it…

Keep regex, fix the syntax!

And lets ensure people actually **adopt** it

```
\b(-?\d+)(?:\.(\d+))?(?:[Ee](-?\d+))?\b
```

```
[#word_boundary
 [capture #integer]
 [0-1 '.' [capture 1+ #digit]]
 [0-1 ['E' | 'e'] [capture #integer]]
#word_boundary]
```

```
\b(-?\d+)(?:\.(\d+))?(?:e(-?\d+))?\b
```

```
[#wb [c #int] [0-1 '.' [c 1+ #d]]
 [0-1 ['E' | 'e'] [c #int]] #wb]
```

```
w{3}\.\w{1,3}\.com
```

```
[3 'w'].[1-3 #token_character].com
```

```
w{3}\.\w{1,3}\.com
```

But you probably meant:

```
[3 'w'].[1-3 [#letter | #digit]].com
```

Or:

```
[3 'w'].[unicode 1-3 [#letter | #digit]].com
```

```
\b(25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
 (25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
 (25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
(25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d)\b
```

```
[#wb #n].[#n].[#n].[#n #wb
  #n=[capture ['25' [0..4] | '24' #d |
                '1' #d #d   |
                [1..9] #d |
                #d]]]
```

```
\b(25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
 (25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
 (25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d).
(25[0..4]|2[0..4]\d|1\d\d|[1..9]\d|\d)\b
```

```
[#wb][#n].[#n].[#n].[#n][#wb][
  #n=[capture [0..255]]
```

```
urlpatterns = patterns('',
    url(r'^tts/(?P<language>\w*)/(?P<phrase>[^/]*)/$',
        tts.views.tts, name='tts'),
    url(r'^tts/(?P<phrase>[^/]*)/$',
        tts.views.tts, name='tts-en'),
)
```

```
import re3

urlpatterns = patterns('',
    url(re3('[#start_linel]tts/[
              capture:language 0+ #token_character]/[
              capture:phrase 0+ not '/']/[#end_line]')),
        tts.views.tts, name='tts'),
    url(re3('[#start_line capture:phrase 0+ not '/']/[#end_line]')),
        tts.views.tts, name='tts-en'),
)
```


Adoption!
Adoption adoption adoption
adoption adoption adoption
adoption adoption adoption
adoption adoption!

How do we convince everyone to switch?

Uses Existing Engines

No risk of incompatibilities

No risk of performance issues

(unless you generate regexes dynamically)

```
import re
INVALID = re.compile(r'\([^)]*\(')
STUFF_IN_PARENS = re.compile(r'\([^)]*\)')
def remove_parentheses(line):
    if INVALID.search(line):
        raise ValueError()
    return STUFF_IN_PARENS.sub('', line)
assert remove_parentheses('a(b)c(d)e') == 'ace'
```

```
import re3
INVALID = re3.compile("([0+ not ')'](")
STUFF_IN_PARENS = re3.compile("([0+ not ')'])")
def remove_parentheses(line):
    if INVALID.search(line):
        raise ValueError()
    return STUFF_IN_PARENS.sub('', line)
assert remove_parentheses('a(b)c(d)e') == 'ace'
```

Available in Python, Javascript, Java, Ruby, Bash

Easy to port (+generic tests)

Try it!

```
$ pip install -e git+git@github.com:SonOfLilit/re2.git#egg=re2
$ echo "Trololo lolo" |
      grep -P "`re2 "[#sl]Tro[0+ #space | 'lo']lo[#el]"`"
```

Interactive Tutorial

Easy to learn

re<->re3 translator

Available as library, command-line tool, **online**

short<->long translator

Type quickly, commit self-documenting code

Same syntax everywhere

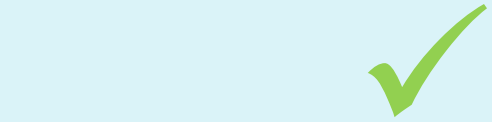
100% platform independent, e.g.

```
$ grep '(\d+)' *.txt
```

```
re.match('(\d+)', text)
```

```
$text ~= /HELLO ((?i)world)/
```

Hello World



(wat)

```
re.match('HELLO ((?i)world)', text)
```

No double escaping

\" ` $ { }

not required

Macros!

e.g.

```
This is a [#trochee #trochee #trochee] regex :-)[
    #trochee=['Robot' | 'Ninja' | 'Pirate' |
              'Doctor' | 'Laser' | 'Monkey']
    [comment 'XKCD856']]
```

DRY

Give meaningful names

Share common subexpressions in libraries

I’m working on this full time

Design	100%
Implementation	80%
Porting	0%
Website, Tutorial	0%
Release Date	**+1.5 weeks**

Please try it

Would you adopt it in your codebase?

```
$ pip install -e git+git@github.com:SonOfLilit/re2.git#egg=re2
$ echo "Trololo lolo" |
      grep -P "`re2 "[#sl]Tro[0+ #space | 'lo']lo[#el]"`"
```

https://github.com/SonOfLilit/re2

Aur Saraf
sonoflilit@gmail.com

Whe “re3”?

Not a new thing, just a new version of regex.
re2 is Google’s open source regex engine.
Name suggestions welcome.

Why not an eDSL?

```
6  LISTING_PATTERN = (begline +
7      label.mode(
8          flag +                              # file type
9          flag*3 + flag*3 + flag*3) +         # owner, group, world perms
10     label.data(
11         somespace + anything +              # links, owner, grp, sz, date
12         somespace +
13         digit*2 + maybe(':') + digit*2 +    # year or hh:mm
14         somespace) +
15     label.file(
16         anybut('->')) +                     # anything before symlink
17     maybe(label.link(
18         somespace + '->' + anything)) +     # possible symlink
19     endline)
```

```
verbal_expression = VerEx()
tester = (verbal_expression.
            start_of_line().
            find('http').
            maybe('s').
            find('://').
            maybe('www.').
            anything_but(' ').
            end_of_line()
)
```

An embedded DSL can’t be copy pasted everywhere, can’t be learned once.

We need **one** new regex language

Is it slower?

```
for i in xrange(10000):
    r = re2.compile("Yo dawg, I heard you like [#word]")
    print r.match(file(str(i) + '.txt', 'rb').read()
```

will only compile once, then run as fast as regular re.

```
for i in xrange(10000):
    r = re2.compile("I got %s problems, but a [#word] ain't one")
    print r.match(file(str(i) + '.txt', 'rb').read()
```

will compile 10K times (but you VERY rarely need 10K different regexes)