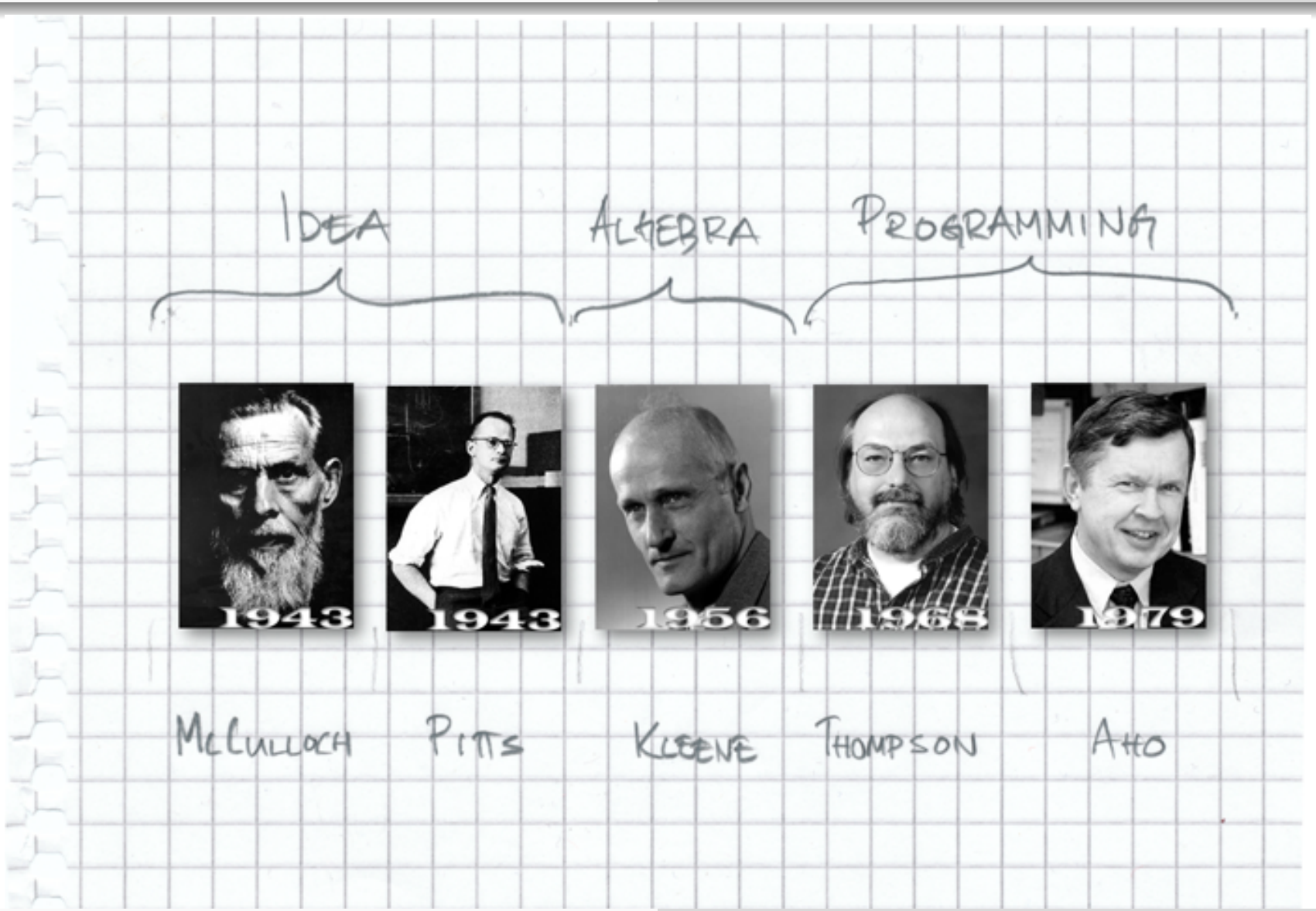
# Introduction to Regular Expressions

Dariusz Śmigiel

San Antonio, 2016-02-25


IDEA
ALGEBRA
PROGRAMMING
1943
1943
1956
1968
1979
McCULLOCH
PITTS
KLEENE
THOMPSON
AHO

Delivering Quality since December 31, 1997

Delivering Quality since December 31, 1997
Release of Python 1.5

Delivering Quality since December 31, 1997
Release of Python 1.5

- Deprecated old module ‘regex‘, based on Perl-style patterns.

Delivering Quality since December 31, 1997
Release of Python 1.5

- Deprecated old module ‘regex‘, based on Perl-style patterns.
- ‘regex‘ finally removed in Python 2.5 (September 19, 2006)

Answer for questions:

Answer for questions:

- "Does this string match the pattern?"
- "Is there a match for the pattern anywhere in this string?"

Answer for questions:

- "Does this string match the pattern?"
- "Is there a match for the pattern anywhere in this string?"
- Replace part of it
- Split into pieces

re is handled as string - there is no special syntax for expressing it (advantage and disadvantage)

re is handled as string - there is no special syntax for expressing it (advantage and disadvantage)
re patterns are compiled into bytecode

re is handled as string - there is no special syntax for expressing it (advantage and disadvantage)
re patterns are compiled into bytecode
re module is a C extension module (like `socket` or `zlib`)

re is handled as string - there is no special syntax for expressing it (advantage and disadvantage)
re patterns are compiled into bytecode
re module is a C extension module (like `socket` or `zlib`)
re language is relatively small and restricted

- not all possible string processing tasks can be done
- some of them can be done, but expression would be very complicated

```
(?:(?:\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])*)?(?:[^()<>@,;:\\".\[\] \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\\]|\\.|(
?:(?:\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])*)?(?:[^()<>@,;:\\".\[\] \0
```

```
00-\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])*)(?:,\s*(
?:(?:[^()<>@,;:\\".\[\] \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\\]|\\.|(?:(?:\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])*)?
(?:[^()<>@,;:\\".\[\] \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])*))*)?;\s*)
```

Perl regex to validate email addresses according to the RFC 822

`http://ex-parrot.com/~pdw/Mail-RFC822-Address`

```
([0-9]+) (\w+ \w+)\n(?P<city>[\w ]+), ([A-Z]{2})
    ↪  (?P<zip>\d{5})
```

```
>>> import re
>>> re.findall('[a-zA-Z0-9]+', 'Search test 03')
['Search', 'test', '03']

>>> import re
>>> regex = re.compile('[a-zA-Z0-9]+')
>>> regex
re.compile('[a-zA-Z0-9]+')
>>> re.findall(regex, 'Search test 02')
['Search', 'test', '02']

>>> import re
>>> regex = re.compile('[a-zA-Z0-9]+')
>>> regex
re.compile('[a-zA-Z0-9]+')
>>> regex.findall('Search test 01')
['Search', 'test', '01']
```

- `re.DEBUG`
- `re.ASCII, re.A`
- `re.IGNORECASE, re.I`
- `re.MULTILINE, re.M`
- `re.DOTALL, re.S`
- `re.VERBOSE, re.X`
- `re.LOCALE, re.L` *Do not use. Deprecated in Py3.5, will be removed in Py3.6*

. ^ $ * + ? { } [ ] \ | ( )

`*` - asterisk. Specifies that previous character can be matched zero or more times.

```
>>> re.findall("ca*t", "ct, cat, caat")
['ct', 'cat', 'caat']
```

`*` - asterisk. Specifies that previous character can be matched zero or more times.

```
>>> re.findall("ca*t", "ct, cat, caat")
['ct', 'cat', 'caat']
```

`+` - plus. Similar to `*`, but requires at least one occurence of character.

```
>>> re.findall("ca+t", "ct, cat, caat")
['cat', 'caat']
```

`*` - asterisk. Specifies that previous character can be matched zero or more times.

```
>>> re.findall("ca*t", "ct, cat, caat")
['ct', 'cat', 'caat']
```

`+` - plus. Similar to `*`, but requires at least one occurence of character.

```
>>> re.findall("ca+t", "ct, cat, caat")
['cat', 'caat']
```

`?` - question mark. Matches either once or zero times

```
>>> re.findall("ca?t", "ct, cat, caat")
['ct', 'cat']
```

`*` - asterisk. Specifies that previous character can be matched zero or more times.

```
>>> re.findall("ca*t", "ct, cat, caat")
['ct', 'cat', 'caat']
```

`+` - plus. Similar to `*`, but requires at least one occurence of character.

```
>>> re.findall("ca+t", "ct, cat, caat")
['cat', 'caat']
```

`?` - question mark. Matches either once or zero times

```
>>> re.findall("ca?t", "ct, cat, caat")
['ct', 'cat']
```

```
>>> re.findall("home-?brew", "homebrew, home-brew")
['homebrew', 'home-brew']
```

`{m,n}` - `m` and `n` are decimal numbers. There must be at least m repetitions, and at most n.

```
>>> re.findall("a/{1,2}b", "ab, a/b, a//b, a///b")
['a/b', 'a//b']
```

`{m,n}` - `m` and `n` are decimal numbers. There must be at least m repetitions, and at most n.

```
>>> re.findall("a/{1,2}b", "ab, a/b, a//b, a///b")
['a/b', 'a//b']
```

`m` and `n` can be ommited. When m ommited, there is zero, when n ommited, upper bound infinity (more precisely, 2 billions)

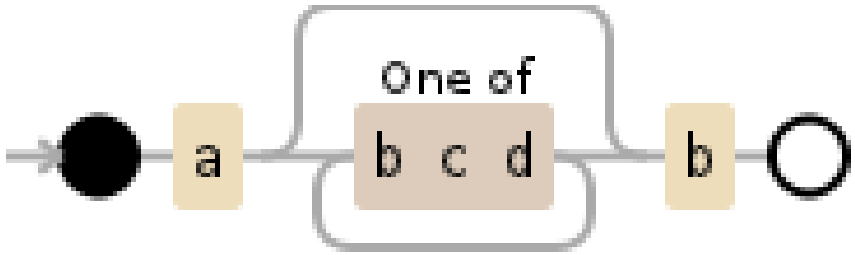
# Equivalents

```
{0,}  ==  "*"
{1,}  ==  "+"
{,1}  ==  {0,1}  ==  "?"
```

`*`, `+`, `?` and `{m,n}` are greedy. Will try to repeat it as many times as possible (re engine can match only 2 billion characters (2GB) – C `int` limitation).

`*`, `+`, `?` and `{m,n}` are greedy. Will try to repeat it as many times as possible (re engine can match only 2 billion characters (2GB) – C `int` limitation).

`a[bcd]*b` - matches `a`, zero or more letters from `bcd`, and ends with `b`




src: `https://www.debuggex.com/r/NT7_HIVhxI_h64zk`

```
re.match("a[bcd]*b", "abcbd")
```

- matches `a`

```
re.match("a[bcd]*b", "abcbd")
```

- matches `a`
- matches `abcbd` to the end of the string

```
re.match("a[bcd]*b", "abcbd")
```

- matches `a`
- matches `abcbd` to the end of the string
- fails, because current position is the end of the string, so cannot match `b`

```
re.match("a[bcd]*b", "abcbd")
```

- matches `a`
- matches `abcbd` to the end of the string
- fails, because current position is the end of the string, so cannot match `b`
- matches `abcb` - one less character

```
re.match("a[bcd]*b", "abcbd")
```

- matches `a`
- matches `abcbd` to the end of the string
- fails, because current position is the end of the string, so cannot match `b`
- matches `abcb` - one less character
- fails, because current position is `d`, so cannot match `b`

`re.match("a[bcd]*b", "abcbd")`

- matches `a`
- matches `abcbd` to the end of the string
- fails, because current position is the end of the string, so cannot match `b`
- matches `abcb` - one less character
- fails, because current position is `d`, so cannot match `b`
- matches `abc`, so `[bcd]*` matches only `bc`

```
re.match("a[bcd]*b", "abcbd")
```

- matches `a`
- matches `abcbd` to the end of the string
- fails, because current position is the end of the string, so cannot match `b`
- matches `abcb` - one less character
- fails, because current position is `d`, so cannot match `b`
- matches `abc`, so `[bcd]*` matches only `bc`
- `abcb`, tries last character `b`, and it's on current position

`re.match("a[bcd]*b", "abcbd")`

- matches `a`
- matches `abcbd` to the end of the string
- fails, because current position is the end of the string, so cannot match `b`
- matches `abcb` - one less character
- fails, because current position is `d`, so cannot match `b`
- matches `abc`, so `[bcd]*` matches only `bc`
- `abcb`, tries last character `b`, and it's on current position
- success

```
>>> re.findall('a[bcd]*b', 'abcbd')
['abcb']
```

`*?, +?, ??, {m,n}?` are non-greedy. Will try to match as few characters as possible.

`*?, +?, ??, {m,n}?` are non-greedy. Will try to match as few characters as possible.

```
>>> import re
>>> text =
     ↪  "<html><head><title>Title</title></head></html>"
```

`*?`, `+?`, `??`, `{m,n}?` are non-greedy. Will try to match as few characters as possible.

```
>>> import re
>>> text =
     ↪  "<html><head><title>Title</title></head></html>"

>>> greedy_regex = re.compile("<.*>")
>>> greedy_regex.findall(text)
['<html><head><title>Title</title></head></html>']
```

`*?, +?, ??, {m,n}?` are non-greedy. Will try to match as few characters as possible.

```
>>> import re
>>> text =
    ↪  "<html><head><title>Title</title></head></html>"

>>> greedy_regex = re.compile("<.*>")
>>> greedy_regex.findall(text)
['<html><head><title>Title</title></head></html>']

>>> non_greedy_regex = re.compile("<.*?>")
>>> non_greedy_regex.findall(text)
['<html>', '<head>', '<title>', '</title>', '</head>',
    ↪  '</html>']
>>>
```

\ - backslash (escape metacharacters)

For matching [ or \ you can use \[ or \\

```
>>> re.findall("\[\]", "Find brackets []")
['[]']
```

\ - backslash (escape metacharacters)
For matching [ or \ you can use \[ or \\

```
>>> re.findall("\[\]", "Find brackets []")
['[]']
```

Some of special sequences beginning with \ express predefined sets of characters: set of digits, letters, everything but whitespace

- re is handled as string

- re is handled as string
- one of `re` metacharacters is \

- re is handled as string
- one of `re` metacharacters is `\`
- backslash for escaping in `re` conflicts with the same purpose in Python

| Characters | Stage |
|---|---|
| `\section` | Text string to be matched |
| `\\section` | Escaped backslash for `re.compile()` |
| `"\\\\section"` | Escaped backslashes for a string literal |

| Characters | Stage |
|---|---|
| `\section` | Text string to be matched |
| `\\section` | Escaped backslash for `re.compile()` |
| `"\\\\section"` | Escaped backslashes for a string literal |

re string needs to be written as `"\\\\"` because regular expression must be `\\` and each must be escaped `\\` inside a regular Python string literal.

| Characters | Stage |
|---|---|
| `\section` | Text string to be matched |
| `\\section` | Escaped backslash for `re.compile()` |
| `"\\\\section"` | Escaped backslashes for a string literal |

re string needs to be written as `"\\\\"` because regular expression must be `\\` and each must be escaped `\\` inside a regular Python string literal.

Solution - raw string

| Regular string | Raw string |
|---|---|
| `"ab*"` | `r"ab*"` |
| `"\\\\section"` | `r"\\section"` |
| `"\\w+\\s+"` | `r"\w+\s+"` |

## match() vs. search()

```
>>> text = 'Your own personal Jesus Someone to hear
    ↪  your prayers Someone who cares Your own
    ↪  personal Jesus'

>>> re.search("Your", text)
<_sre.SRE_Match object; span=(0, 4), match='Your'>

>>> re.match("Your", text)
<_sre.SRE_Match object; span=(0, 4), match='Your'>

>>> re.search("Jesus", text)
<_sre.SRE_Match object; span=(18, 23), match='Jesus'>

>>> re.match("Jesus", text)
>>>
```

```
>>> date = "10 December 2015"
>>> matched = re.match("(\d+) (\w+) (\d{4})", date)
>>> matched.groups()
('10', 'December', '2015')
>>> matched.group(1)
'10'
>>> matched.group(2)
'December'
>>> matched.group(3)
'2015'
>>>
```

```
>>> date = "10 December 2015"
>>> matched = re.match("(?P<day>\d+) (?P<month>\w+)
    ↪ (?P<year>\d{4})", date)
>>> matched.groups()
('10', 'December', '2015')
>>> matched.group('day')
'10'
>>> matched.group('month')
'December'
>>> matched.group('year')
'2015'
>>> matched.groupdict()
{'month': 'December', 'day': '10', 'year': '2015'}
>>>
```

**Language feature comparison (part 1)**

| | "+" quantifier | Negated character classes | Non-greedy quantifiers[Note 1] | Shy groups[Note 2] | Recursion | Look-ahead | Look-behind | Backreferences[Note 3] | >9 indexable captures |
|---|---|---|---|---|---|---|---|---|---|
| Boost.Regex | Yes | Yes | Yes | Yes | Yes[Note 4] | Yes | Yes | Yes | Yes |
| Boost.Xpressive | Yes | Yes | Yes | Yes | Yes[Note 5] | Yes | Yes | Yes | Yes |
| CL-PPCRE | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| EmEditor | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | No |
| FREJ | No[Note 6] | No | Some[Note 6] | Yes | No | No | No | Yes | Yes |
| GLib/GRegex | Yes | ? | Yes | ? | No | ? | ? | ? | ? |
| GNU grep | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | ? |
| Haskell | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| ICU Regex | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| Java | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| JavaScript (ECMAScript) | Yes | Yes | Yes | Yes | No | Yes | No | Yes | Yes |
| JGsoft | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| Lua | Yes | Yes | Yes | No | No | No | No | Yes | No |
| .NET | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| OCaml | Yes | Yes | No | No | No | No | No | Yes | No |
| OmniOutliner 3.6.2 | Yes | Yes | Yes | No | No | No | No | ? | ? |
| PCRE | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Perl | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| PHP | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Python | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| Qt/QRegExp | Yes | Yes | Yes | Yes | No | Yes | No | Yes | Yes |
| R[Note 7] | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |

**Language feature comparison (part 2)**

| | Directives[Note 1] | Conditionals | Atomic groups[Note 2] | Named capture[Note 3] | Comments | Embedded code | Unicode property support [1] | Balancing groups[Note 4] | Variable-length look-behinds[Note 5] |
|---|---|---|---|---|---|---|---|---|---|
| **Boost.Regex** | Yes | Yes | Yes | Yes | Yes | No | Some[Note 6] | No | No |
| **Boost.Xpressive** | Yes | No | Yes | Yes | Yes | No | No | No | No |
| **CL-PPCRE** | Yes | Yes | Yes | Yes | Yes | Yes | Some[Note 6] | No | No |
| **EmEditor** | Yes | Yes | ? | ? | Yes | No | ? | No | No |
| **FREJ** | No | No | Yes | Yes | Yes | No | ? | No | No |
| **GLib/GRegex** | Yes | Yes | Yes | Yes | Yes | No | Some[Note 6] | No | No |
| **GNU grep** | Yes | Yes | ? | Yes | Yes | No | No | No | No |
| **Haskell** | ? | ? | ? | ? | ? | No | No | No | No |
| **ICU Regex** | Yes | No | Yes | No | Yes | No | Yes | No | No |
| **Java** | Yes | No | Yes | Yes[Note 7] | Yes | No | Some[Note 6] | No | No |
| **JavaScript (ECMAScript)** | No | No | No | No | No | No | No | No | No |
| **JGsoft** | Yes | Yes | Yes | Yes | Yes | No | Some[Note 6] | No | Yes |
| **Lua** | No | No | No | No | No | No | No | No | No |
| **.NET** | Yes | Yes | Yes | Yes | Yes | No | Some[Note 6] | Yes | Yes |
| **OCaml** | No | No | No | No | No | No | No | No | No |
| **OmniOutliner 3.6.2** | ? | ? | ? | ? | No | No | ? | No | No |
| **PCRE** | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | No |
| **Perl** | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | No |
| **PHP** | Yes | Yes | Yes | Yes | Yes | No | No | No | No |
| **Python** | Yes | Yes | No | Yes | Yes | No | No | No | No |
| **Qt/QRegExp** | No | No | No | No | No | No | No | No | No |
| **RE2** | Yes | No | ? | Yes | No | No | Some[Note 6] | No | No |

```
([0-9]+) (\w+\ \w+)\n(?P<city>[\w ]+), ([A-Z]{2})
    ↪  (?P<zip>\d{5})
```

```
def verify_regex(text):
    regex = """
    ([0-9]+)      # Match any recurring numbers
    \             # One whitespace
    (\w+ \w+)     # Match street
    \n            # New line
    (?P<city>[\w ]+) # Named group for City
    ,             # Comma :)
    \             # One whitespace
    ([A-Z]{2})    # Two capital letters, matching state
    \             # One whitespace
    (?P<zip>\d{5}) # Named group for matching ZIP code
    """
    re_compile = re.compile(regex, re.X)
    matched = re_compile.match(text)
    print(matched.groups())
```

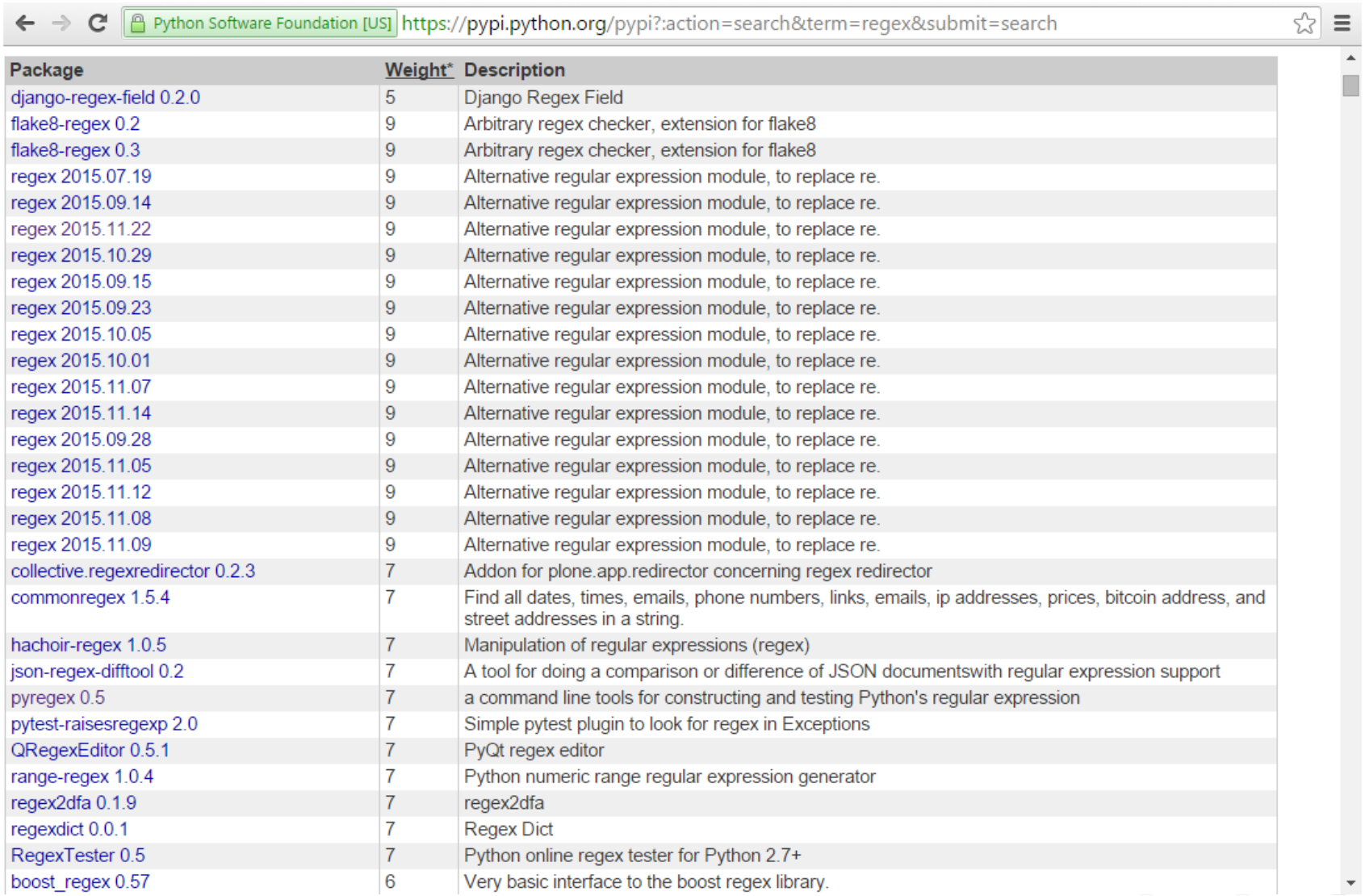
```
>>> address = """1 Fanatical Place
San Antonio, TX 78218"""
>>> verify_regex(address)
('1', 'Fanatical Place', 'San Antonio', 'TX', '78218')
>>>
```

```
def parse_volume_type_from_filename(self,
    ↪  emcConfigFile):
    """Parse the volume type from the file (if it
        ↪  exists).

    :param emcConfigFile: the EMC configuration file
    :returns: volumeTypeName - the volume type name
    """
    volumeTypeName = None

    m =
        ↪  re.search('/etc/cinder/cinder_emc_config_(.+?).xml
        ↪  emcConfigFile)
    if m:
        volumeTypeName = m.group(1)

    return volumeTypeName
"cinder/cinder/volume/drivers/emc/emc_vmax_utils.py"
```




| Package | Weight* | Description |
|---|---|---|
| django-regex-field 0.2.0 | 5 | Django Regex Field |
| flake8-regex 0.2 | 9 | Arbitrary regex checker, extension for flake8 |
| flake8-regex 0.3 | 9 | Arbitrary regex checker, extension for flake8 |
| regex 2015.07.19 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.09.14 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.11.22 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.10.29 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.09.15 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.09.23 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.10.05 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.10.01 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.11.07 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.11.14 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.09.28 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.11.05 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.11.12 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.11.08 | 9 | Alternative regular expression module, to replace re. |
| regex 2015.11.09 | 9 | Alternative regular expression module, to replace re. |
| collective.regexredirector 0.2.3 | 7 | Addon for plone.app.redirector concerning regex redirector |
| commonregex 1.5.4 | 7 | Find all dates, times, emails, phone numbers, links, emails, ip addresses, prices, bitcoin address, and street addresses in a string. |
| hachoir-regex 1.0.5 | 7 | Manipulation of regular expressions (regex) |
| json-regex-difftool 0.2 | 7 | A tool for doing a comparison or difference of JSON documentswith regular expression support |
| pyregex 0.5 | 7 | a command line tools for constructing and testing Python's regular expression |
| pytest-raisesregexp 2.0 | 7 | Simple pytest plugin to look for regex in Exceptions |
| QRegexEditor 0.5.1 | 7 | PyQt regex editor |
| range-regex 1.0.4 | 7 | Python numeric range regular expression generator |
| regex2dfa 0.1.9 | 7 | regex2dfa |
| regexdict 0.0.1 | 7 | Regex Dict |
| RegexTester 0.5 | 7 | Python online regex tester for Python 2.7+ |
| boost_regex 0.57 | 6 | Very basic interface to the boost regex library. |

Wrapper for Google RE2 engine
Prerequisites for re2 in Python:

- RE2 library from Google
- Python development headers
- build environment with g++

Released in April 2015. In development since 2010.
Features

- `fullmatch` works like `match` but anchors the match at both the start and the end
- `test_(search|match|fullmatch)` methods that work like `(search|match|fullmatch)` but only returns `True` or `False`

## Missing Features

- no substitution methods
- no flags
- no `split`, `findall`, `finditer`
- no top-level functions like `search` or `match`; just use compile
- no compile cache
- no `lastindex` or `lastgroup` on `Match` objects

Initially fork of Facebook version, later rewritten from scratch.
Features

- Backward compatibility

```
try:
    import re2 as re
except ImportError:
    import re
```

`re2` will automatically fall back to the original `re` module if there is a regex that it cannot handle: `re2` doesn't handle lookahead assertions `(?=...)`.

Missing Features/Flaws

- Unicode support: module supports UTF8, so automatically encodes and decodes any unicode string.

Total runs: 100

| Test | `re` time (s) | `re2` time (s) | % time |
|---|---|---|---|
| Findall URI or Emails | 348.380 | 8.139 | 2.34% |
| Replace wikilinks | 3.659 | 0.812 | 22.19% |
| Remove wikilinks | 3.553 | 0.273 | 7.70% |

src: `https://github.com/axiak/pyre2/blob/master/tests/performance.py`

- Regular Expression HOWTO: https://docs.python.org/2/howto/regex.html
- Python Docs: Library re: https://docs.python.org/2/library/re.html
- Google for Education. Python Regular Expressions: https://developers.google.com/edu/python/regular-expressions?hl=en
- Regex Debugger: https://regex101.com/
- Debuggex: https://www.debuggex.com/
- Core Python Applications programming: Regular expressions: http://www.informit.com/articles/article.aspx?p=1707750&seqNum=2
- Brief history by Staffan Noteberg: http://blog.staffannoteberg.com/
- Regular Expression Matching Can Be Simple And Fast https://swtch.com/~rsc/regexp/regexp1.html

- Google RE2: https://github.com/google/re2
- Facebook re2: https://github.com/facebook/pyre2
- Axiak re2: https://github.com/axiak/pyre2
- re2 on PyPI: https://pypi.python.org/pypi/re2

# Thank you

Questions?