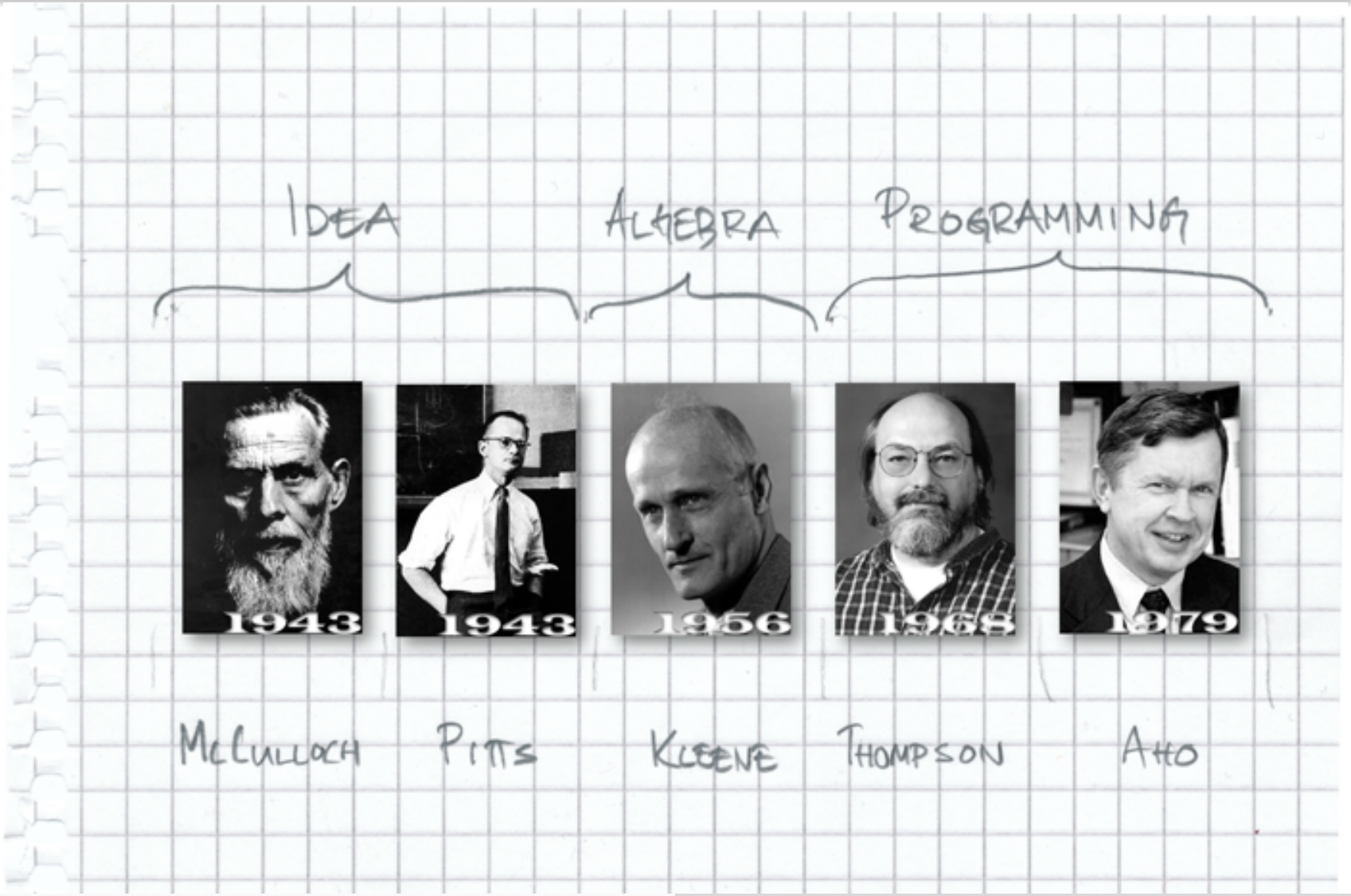
# HowTo re

Dariusz Śmigiel

PyCon PL 2015


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:

- "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

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

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

## [ ] - class

```
>>> import re
>>> re.findall("[def]", "abcdefghi")
['d', 'e', 'f']
>>> re.findall("[d-f]", "abcdefghi")
['d', 'e', 'f']
>>>
```

## [ ] - class

```
>>> import re
>>> re.findall("[def]", "abcdefghi")
['d', 'e', 'f']
>>> re.findall("[d-f]", "abcdefghi")
['d', 'e', 'f']
>>>
```

## Metacharacters are not active inside class

```
>>> re.findall("[d-f$]", "abcdefg$hi")
['d', 'e', 'f', '$']
```

## ^ - complementing set

```
>>> import re
>>> re.findall("[^5]", "abc 456 xyz")
['a', 'b', 'c', ' ', '4', '6', ' ', 'x', 'y', 'z']
>>>
```

. - dot. Matches anything except a newline character

```
>>> import re
>>> regex = re.compile(".")
>>> regex.findall("string")
['s', 't', 'r', 'i', 'n', 'g']
>>>
```

. - dot. Matches anything except a newline character

```
>>> import re
>>> regex = re.compile(".")
>>> regex.findall("string")
['s', 't', 'r', 'i', 'n', 'g']
>>>
```

There is compilation flag, which changes default behavior.

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

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

`*` - star. 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']
```

`*` - star. 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']
```

`*` - star. 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']
```

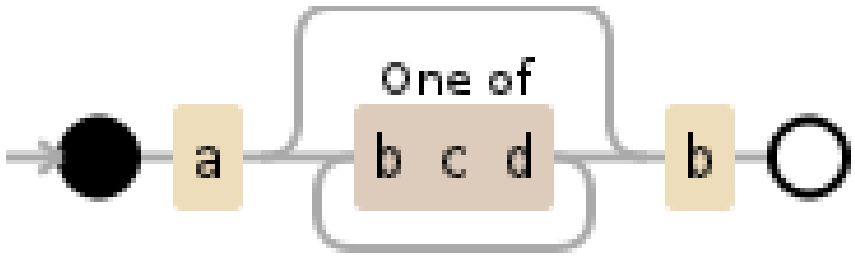
`m` and `n` can be ommited. When m ommited, there is zero, when n ommited, upper bound infinity (more precisely, 2 billions)

```
{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

- `\d` - any decimal digit, equivalent of `[0-9]`
- `\D` - everything but decimal digit, equivalent of `[^0-9]`

```
>>> re.findall("\d", "abc789xyz")
['7', '8', '9']
>>> re.findall("[0-9]", "abc789xyz")
['7', '8', '9']
```

- `\d` - any decimal digit, equivalent of `[0-9]`
- `\D` - everything but decimal digit, equivalent of `[^0-9]`

```
>>> re.findall("\d", "abc789xyz")
['7', '8', '9']
>>> re.findall("[0-9]", "abc789xyz")
['7', '8', '9']

>>> re.findall("\D", "abc789xyz")
['a', 'b', 'c', 'x', 'y', 'z']
>>> re.findall("[^0-9]", "abc789xyz")
['a', 'b', 'c', 'x', 'y', 'z']
```

- `\w` - any alphanumeric: `[a-zA-Z0-9_]`
- `\W` - any non-alphanumeric: `[^a-zA-Z0-9_]`

```
>>> re.findall('\w+', 'abc 789 xyz')
['abc', '789', 'xyz']
>>> re.findall('[a-zA-Z0-9_]+', 'abc 789 xyz')
['abc', '789', 'xyz']
```

- \w - any alphanumeric: [a-zA-Z0-9_]
- \W - any non-alphanumeric: [^a-zA-Z0-9_]

```
>>> re.findall('\w+', 'abc 789 xyz')
['abc', '789', 'xyz']
>>> re.findall('[a-zA-Z0-9_]+', 'abc 789 xyz')
['abc', '789', 'xyz']

>>> re.findall('\W+', 'abc 789 xyz')
[' ', ' ']
>>> re.findall('[^a-zA-Z0-9_]+', 'abc 789 xyz')
[' ', ' ']
```

- `\s` - any whitespace character: `[ \t\n\r\f\v]` (space, tab (ASCII 0x09), newline (0x0A), return (0x0D), form feed - page break(0x0C), vertical tab (0x0B))
- `\S` - any non-whitespace character: `[^ \t\n\r\f\v]`

- `\s` - any whitespace character: `[ \t\n\r\f\v]` (space, tab (ASCII 0x09), newline (0x0A), return (0x0D), form feed - page break(0x0C), vertical tab (0x0B))
- `\S` - any non-whitespace character: `[^ \t\n\r\f\v]`
- NOTE! Remember that Windows text files use `\r\n` to terminate lines, while UNIX text files use `\n`.

```
>>> text = "line,\nwith \ttab,\vvertical,\rreturn and
    ↪  \nnewlines."
>>> print(text)
line,
with    tab,
return and  vertical,
newlines.
```

```
>>> text = "line,\nwith \ttab,\vvertical,\rreturn and
    ↪  \nnewlines."
>>> print(text)
line,
with    tab,
return and  vertical,
newlines.

>>> re.findall("\s+", text)
['\n', ' \t', '\x0b', '\r', ' ', ' \n']
```

```
>>> text = "line,\nwith \ttab,\vvertical,\rreturn and
    ↪  \nnewlines."
>>> print(text)
line,
with    tab,
return and  vertical,
newlines.

>>> re.findall("\s+", text)
['\n', ' \t', '\x0b', '\r', ' ', ' \n']

>>> re.findall("\S+", text)
['line,', 'with', 'tab,', 'vertical,', 'return', 'and',
    ↪  'newlines.']
```

- 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+"` |

```
>>> latex = """
... \begin{document}
... \section{History}
... \subsection{Origins}
... \begin{frame}
... Content
... \end{frame}
... \end{document}
... """
```

```
>>> latex = """
... \begin{document}
... \section{History}
... \subsection{Origins}
... \begin{frame}
... Content
... \end{frame}
... \end{document}
... """

>>> latex
'\n\x08egin{document}\n\\section{History}\n\\subsection{Origin
```

```
>>> latex = """
... \begin{document}
... \section{History}
... \subsection{Origins}
... \begin{frame}
... Content
... \end{frame}
... \end{document}
... """

>>> latex
'\n\x08egin{document}\n\\section{History}\n\\subsection{Origin

>>> print(re.findall(r"\\section{.*}", latex))
['\\section{History}']
>>> print(re.findall(r"\\section{.*}", latex)[0])
\section{History}
```

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

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

>>> 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']

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

## re.DEBUG

```
>>> import re
>>> regex = re.compile('[a-z]', re.DEBUG)
in
  range (97, 122)
>>>
```

`re.ASCII, re.A`

`\xa0` - non-breaking space

```
>>> import re
>>> regex = re.compile("\s+")
>>> regex.findall("\xa0 ha")
['\xa0 ']
```

## `re.ASCII, re.A`

`\xa0` - non-breaking space

```
>>> import re
>>> regex = re.compile("\s+")
>>> regex.findall("\xa0 ha")
['\xa0 ']

>>> ascii_regex = re.compile("\s+", re.ASCII)
>>> ascii_regex.findall("\xa0 ha")
[' ']
>>>
```

## re.IGNORECASE, re.I

```
>>> import re
>>> text = "CamelCase CAPITAL and lower WoRd"
>>> regex = re.compile("[a-z]+")
>>> regex.findall(text)
['amel', 'ase', 'and', 'lower', 'o', 'd']
```

## re.IGNORECASE, re.I

```
>>> import re
>>> text = "CamelCase CAPITAL and lower WoRd"
>>> regex = re.compile("[a-z]+")
>>> regex.findall(text)
['amel', 'ase', 'and', 'lower', 'o', 'd']

>>> ignorecase_regex = re.compile("[a-z]+", re.I)
>>> ignorecase_regex.findall(text)
['CamelCase', 'CAPITAL', 'and', 'lower', 'WoRd']
>>>
```

## re.MULTILINE, re.M

```
>>> import re
>>> text = """From the beginning,
... in the middle,
... and at the end."""
```

## re.MULTILINE, re.M

```
>>> import re
>>> text = """From the beginning,
... in the middle,
... and at the end."""

>>> regex = re.compile("^[a-zA-Z]+")
>>> regex.findall(text)
['From']
```

## re.MULTILINE, re.M

```
>>> import re
>>> text = """From the beginning,
... in the middle,
... and at the end."""

>>> regex = re.compile("^[a-zA-Z]+")
>>> regex.findall(text)
['From']

>>> multiline_regex = re.compile("^[a-zA-Z]+", re.M)
>>> multiline_regex.findall(text)
['From', 'in', 'and']
>>>
```

## re.DOTALL, re.S

```
>>> import re
>>> text = """From the beginning,
... in the middle,
... and at the end."""
```

## re.DOTALL, re.S

```
>>> import re
>>> text = """From the beginning,
... in the middle,
... and at the end."""

>>> regex = re.compile(".+")
>>> regex.findall(text)
['From the beginning,', 'in the middle,', 'and at the
    ↪  end.']
```

## re.DOTALL, re.S

```
>>> import re
>>> text = """From the beginning,
... in the middle,
... and at the end."""

>>> regex = re.compile(".+")
>>> regex.findall(text)
['From the beginning,', 'in the middle,', 'and at the
    ↪  end.']

>>> dotall_regex = re.compile(".+", re.S)
>>> dotall_regex.findall(text)
['From the beginning,\nin the middle,\nand at the end.']
>>>
```

## re.VERBOSE, re.X

```
>>> import re
>>> numbers = "127.2, 15.30, 73"
>>> regex = re.compile(r"\d+\.?\d*")
>>> regex.findall(numbers)
['127.2', '15.30', '73']
```

## re.VERBOSE, re.X

```
>>> import re
>>> numbers = "127.2, 15.30, 73"
>>> regex = re.compile(r"\d+\.?\d*")
>>> regex.findall(numbers)
['127.2', '15.30', '73']

>>> verbose_regex = re.compile(r"""
        \d +  # the integral part
        \. ?  # the decimal point
        \d *  # some fractional digits""", re.X)
>>> verbose_regex.findall(numbers)
['127.2', '15.30', '73']
>>>
```

`re.LOCALE, re.L`

Make `\w`, `\W`, `\b`, and `\B`, dependent on the current locale instead of the Unicode database.

*Do not use.*

*Deprecated in Python 3.5, will be removed in version 3.6*

| - "or" operator

```
>>> re.findall("No|Yes", "Yes and No")
['Yes', 'No']
```

## | - "or" operator

```
>>> re.findall("No|Yes", "Yes and No")
['Yes', 'No']

>>> re.findall("Yes|No", "Yes|No")
['Yes', 'No']
>>> re.findall("Yes\|No", "Yes|No")
['Yes|No']
>>>
```

## ^, \A - beginning of lines

```
>>> text = """Your own personal Jesus
... Someone to hear your prayers
... Someone who cares
... Your own personal Jesus
... Someone to hear your prayers
... Someone who's there"""
```

## ^, \A - beginning of lines

```
>>> text = """Your own personal Jesus
... Someone to hear your prayers
... Someone who cares
... Your own personal Jesus
... Someone to hear your prayers
... Someone who's there"""

>>> re.findall("^Your", text)
['Your']
>>> re.findall("\AYour", text)
['Your']
```

## ^, \A - beginning of lines

```
>>> text = """Your own personal Jesus
... Someone to hear your prayers
... Someone who cares
... Your own personal Jesus
... Someone to hear your prayers
... Someone who's there"""

>>> re.findall("^Your", text)
['Your']
>>> re.findall("\AYour", text)
['Your']

>>> re.findall("^Your", text, re.M)
['Your', 'Your']
>>> re.findall("\AYour", text, re.M)
['Your']
```

## `$`, `\Z` - end of lines

```
>>> text = """Your own personal Jesus
... Someone to hear your prayers
... Someone who cares
... Your own personal Jesus"""
```

## `$`, `\Z` - end of lines

```
>>> text = """Your own personal Jesus
... Someone to hear your prayers
... Someone who cares
... Your own personal Jesus"""

>>> re.findall("Jesus$", text)
['Jesus']
>>> re.findall("Jesus\Z", text)
['Jesus']
```

## `$`, `\Z` - end of lines

```
>>> text = """Your own personal Jesus
... Someone to hear your prayers
... Someone who cares
... Your own personal Jesus"""

>>> re.findall("Jesus$", text)
['Jesus']
>>> re.findall("Jesus\Z", text)
['Jesus']

>>> re.findall("Jesus$", text, re.M)
['Jesus', 'Jesus']
>>> re.findall("Jesus\Z", text, re.M)
['Jesus']
```

## \b, \B - word boundaries

```
>>> text = "People in class heard that Pluto should be
    ↪  reclassified, because it is no longer a planet."
```

## \b, \B - word boundaries

```
>>> text = "People in class heard that Pluto should be
    ↪  reclassified, because it is no longer a planet."

>>> re.sub("class", "room", text)
'People in room heard that Pluto should be reroomified,
    ↪  because it is no longer a planet.'
```

## \b, \B - word boundaries

```
>>> text = "People in class heard that Pluto should be
    ↪  reclassified, because it is no longer a planet."

>>> re.sub("class", "room", text)
'People in room heard that Pluto should be reroomified,
    ↪  because it is no longer a planet.'

>>> re.sub(r"\bclass\b", "room", text)
'People in room heard that Pluto should be
    ↪  reclassified, because it is no longer a planet.'
```

## \b, \B - word boundaries

```
>>> text = "People in class heard that Pluto should be
    ↪  reclassified, because it is no longer a planet."

>>> re.sub("class", "room", text)
'People in room heard that Pluto should be reroomified,
    ↪  because it is no longer a planet.'

>>> re.sub(r"\bclass\b", "room", text)
'People in room heard that Pluto should be
    ↪  reclassified, because it is no longer a planet.'

>>> re.sub(r"\Bclass\B", "qual", text)
'People in class heard that Pluto should be
    ↪  requalified, because it is no longer a planet.'
```

## match() vs. search()

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

## 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'>
```

## 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'>
```

## 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'>
```

## 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)
>>>
```

## `findall()` vs. `finditer()`

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

## findall() vs. finditer()

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

>>> output_findall = re.findall("Someone", text)
>>> output_finditer = re.finditer("Someone", text)
```

## findall() vs. finditer()

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

>>> output_findall = re.findall("Someone", text)
>>> output_finditer = re.finditer("Someone", text)

>>> type(output_findall)
<class 'list'>
>>> type(output_finditer)
<class 'callable_iterator'>
```

```
>>> output_findall
['Someone', 'Someone']
>>> output_finditer
<callable_iterator object at 0x7f69ffd267b8>
```

```
>>> output_findall
['Someone', 'Someone']
>>> output_finditer
<callable_iterator object at 0x7f69ffd267b8>

>>> list(output_finditer)
[<_sre.SRE_Match object; span=(24, 31),
    ↪  match='Someone'>, <_sre.SRE_Match object;
    ↪  span=(53, 60), match='Someone'>]
>>>
```

```
>>> matched = re.match("\d{0,2}-\d{0,3}", "88-299")
>>> matched
<_sre.SRE_Match object; span=(0, 6), match='88-299'>
>>> if matched:
...     # do something
...     pass
...
```

```
>>> matched = re.match("\d{0,2}-\d{0,3}", "88-299")
>>> matched
<_sre.SRE_Match object; span=(0, 6), match='88-299'>
>>> if matched:
...     # do something
...     pass
...

>>> non_matched = re.match("(\d?)-(\d{0,3})", "88-299")
>>> non_matched
>>> non_matched == None
True
>>>
```

## start() and end()

```
>>> text = "Soft Kitty, warm Kitty"
>>> matched = re.search("Kitty", text)
>>> matched.start()
5
>>> matched.end()
10
```

## match.re and match.string

```
>>> matched = re.match(r"(\w+)@(\w+\.\w+)",
    ↪  "login@server.com")
```

## match.re and match.string

```
>>> matched = re.match(r"(\w+)@(\w+\.\w+)",
    ↪  "login@server.com")

>>> matched.re
re.compile('(\\w+)@(\\w+\\.\\w+)')
>>> matched.string
'login@server.com'
>>>
```

## Split

```
>>> text = "Oh, what a day. What a lovely day!"
>>> re.split("\W+", text)
['Oh', 'what', 'a', 'day', 'What', 'a', 'lovely',
    ↪  'day', '']
```

## Split

```
>>> text = "Oh, what a day. What a lovely day!"
>>> re.split("\W+", text)
['Oh', 'what', 'a', 'day', 'What', 'a', 'lovely',
    ↪  'day', '']

>>> re.split("(\W+)", text)
['Oh', ', ', 'what', ' ', 'a', ' ', 'day', '. ',
    ↪  'What', ' ', 'a', ' ', 'lovely', ' ', 'day',
    ↪  '!', '']
>>>
```

## Search and replace - `sub()` and `subn()`

`sub()` *is deprecated since Python 3.5 and will be removed in 3.6*

```
>>> pattern = r"\bBar\b"
>>> replacement = "Baz"
>>> string = "Foo Bar"
```

## Search and replace - `sub()` and `subn()`

`sub()` *is deprecated since Python 3.5 and will be removed in 3.6*

```
>>> pattern = r"\bBar\b"
>>> replacement = "Baz"
>>> string = "Foo Bar"

>>> new_string, number_of_subs_made = re.subn(pattern,
    ↪  replacement, string)
>>> new_string, number_of_subs_made
('Foo Baz', 1)
>>>
```

```
>>> date = "15 October 2015"
>>> matched = re.match("(\d+) (\w+) (\d{4})", date)
>>> matched.groups()
('15', 'October', '2015')
>>> matched.group(1)
'15'
>>> matched.group(2)
'October'
>>> matched.group(3)
'2015'
>>>
```

## Named groups

```
>>> date = "15 October 2015"
>>> matched = re.match("(?P<day>\d+) (?P<month>\w+)
    ↪  (?P<year>\d{4})", date)
>>> matched.groups()
('15', 'October', '2015')
>>> matched.group('day')
'15'
>>> matched.group('month')
'October'
>>> matched.group('year')
'2015'
>>>
```

```
>>> m = re.search(r'(?P<quote>")(.*)(?P=quote)', 'This
    ↪  is "quote"')
>>> m
<_sre.SRE_Match object; span=(8, 15), match='"quote"'>
>>> m.groups()
('"', 'quote')
```

## Positive and negative lookahead assertions

```
>>> singer = "Michael Jackson"
>>> player = "Michael Jordan"
>>> player_pattern = "Michael (?=Jordan)"
>>> non_player_pattern = "Michael (?!Jordan)"
```

## Positive and negative lookahead assertions

```
>>> singer = "Michael Jackson"
>>> player = "Michael Jordan"
>>> player_pattern = "Michael (?=Jordan)"
>>> non_player_pattern = "Michael (?!Jordan)"

>>> re.match(player_pattern, player)
<_sre.SRE_Match object; span=(0, 8), match='Michael '>
>>> re.match(non_player_pattern, singer)
<_sre.SRE_Match object; span=(0, 8), match='Michael '>
```

## Positive and negative lookahead assertions

```
>>> singer = "Michael Jackson"
>>> player = "Michael Jordan"
>>> player_pattern = "Michael (?=Jordan)"
>>> non_player_pattern = "Michael (?!Jordan)"

>>> re.match(player_pattern, player)
<_sre.SRE_Match object; span=(0, 8), match='Michael '>
>>> re.match(non_player_pattern, singer)
<_sre.SRE_Match object; span=(0, 8), match='Michael '>

>>> re.match(player_pattern, singer)
>>> re.match(non_player_pattern, player)
>>>
```

## Positive and negative lookbehind assertions

```
>>> string = """
... def function():
...     return function()
"""
>>> re.search("(?<=def )function", string)
<_sre.SRE_Match object; span=(4, 12), match='function'>
>>> re.search("(?<!def )function", string)
<_sre.SRE_Match object; span=(27, 35), match='function'>
>>>
```

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 |
| RE2 | Yes | Yes | Yes | Yes | No | No | No | No | Yes |
| Ruby | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| TRE | Yes | Yes | Yes | Yes | No | No | No | Yes | No |
| Vim 7.4b.000 (2013-07-28) [±] | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | No |
| RGX | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| Tcl | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| TRegExpr | Yes | ? | Yes | ? | ? | ? | ? | ? | ? |
| XRegExp | Yes | Yes | Yes | Yes | No | Yes | No | 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 |
| Ruby | Yes | No | Yes | Yes | Yes | Yes | Some[Note 6] | No | No |
| Tcl | Yes | No | Yes | No | Yes | No | Yes | No | No |
| TRE | Yes | No | No | No | Yes | No | ? | No | No |
| Vim | Yes | No | Yes | No | No | No | No | No | Yes |
| RGX | Yes | Yes | Yes | Yes | Yes | No | Yes | No | No |
| XRegExp | Leading only | No | No | Yes | Yes | No | Yes | No | No |

- 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/`