

:mod:`re` --- Regular expression operations

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1); [backlink](#)

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 4)

Unknown directive type "module".

```
.. module:: re
   :synopsis: Regular expression operations.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 7)

Unknown directive type "moduleauthor".

```
.. moduleauthor:: Fredrik Lundh <fredrik@pythonware.com>
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 8)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Andrew M. Kuchling <amk@amk.ca>
```

Source code: `:source:`Lib/re.py``

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 10); [backlink](#)

Unknown interpreted text role "source".

This module provides regular expression matching operations similar to those found in Perl.

Both patterns and strings to be searched can be Unicode strings (`:class:`str``) as well as 8-bit strings (`:class:`bytes``). However, Unicode strings and 8-bit strings cannot be mixed: that is, you cannot match a Unicode string with a byte pattern or vice-versa; similarly, when asking for a substitution, the replacement string must be of the same type as both the pattern and the search string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 17); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 17); [backlink](#)

Unknown interpreted text role "class".

Regular expressions use the backslash character (`'\'`) to indicate special forms or to allow special characters to be used without invoking their special meaning. This collides with Python's usage of the same character for the same purpose in string literals; for example, to match a literal backslash, one might have to write `'\\\\'` as the pattern string, because the regular expression must be `\\`, and each backslash must be expressed as `\\` inside a regular Python string literal. Also, please note that any invalid escape sequences in Python's usage of the backslash in string literals now generate a `:exc:`DeprecationWarning`` and in the future this will become a `:exc:`SyntaxError``. This behaviour will happen even if it is a valid escape sequence for a regular expression.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 24); [backlink](#)

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 24); [backlink](#)

Unknown interpreted text role "exc".

The solution is to use Python's raw string notation for regular expression patterns; backslashes are not handled in any special way in a string literal prefixed with 'r'. So `r"\n"` is a two-character string containing backslash and 'n', while `"\n"` is a one-character string containing a newline. Usually patterns will be expressed in Python code using this raw string notation.

It is important to note that most regular expression operations are available as module-level functions and methods on [ref`compiled regular expressions <re-objects>](#). The functions are shortcuts that don't require you to compile a regex object first, but miss some fine-tuning parameters.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 43); [backlink](#)

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 49)

Unknown directive type "seealso".

```
.. seealso::
```

```
The third-party `regex <https://pypi.org/project/regex/>`_ module,
which has an API compatible with the standard library :mod:`re` module,
but offers additional functionality and a more thorough Unicode support.
```

Regular Expression Syntax

A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

Regular expressions can be concatenated to form new regular expressions; if *A* and *B* are both regular expressions, then *AB* is also a regular expression. In general, if a string *p* matches *A* and another string *q* matches *B*, the string *pq* will match *AB*. This holds unless *A* or *B* contain low precedence operations; boundary conditions between *A* and *B*; or have numbered group references. Thus, complex expressions can easily be constructed from simpler primitive expressions like the ones described here. For details of the theory and implementation of regular expressions, consult the Friedl book [Frie09], or almost any textbook about compiler construction.

A brief explanation of the format of regular expressions follows. For further information and a gentler presentation, consult the [ref`regex-howto`](#).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 76); [backlink](#)

Unknown interpreted text role "ref".

Regular expressions can contain both special and ordinary characters. Most ordinary characters, like 'A', 'a', or '0', are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so `last` matches the string 'last'. (In the rest of this section, we'll write RE's in this special style, usually without quotes, and strings to be matched 'in single quotes'.)

Some characters, like '|' or '(', are special. Special characters either stand for classes of ordinary characters, or affect how the regular expressions around them are interpreted.

Repetition operators or quantifiers (`*`, `+`, `?`, `{m,n}`, etc) cannot be directly nested. This avoids ambiguity with the non-greedy modifier suffix `?`, and with other modifiers in other implementations. To apply a second repetition to an inner repetition, parentheses may be used. For example, the expression `(?:a{6})*` matches any multiple of six 'a' characters.

The special characters are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 99)

Unknown directive type "index".

```
.. index:: single: . (dot); in regular expressions
```

(Dot.) In the default mode, this matches any character except a newline. If the `:const:'DOTALL'` flag has been specified, this matches any character including a newline.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 102);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 106)

Unknown directive type "index".

```
.. index:: single: ^ (caret); in regular expressions
```

(Caret.) Matches the start of the string, and in `:const:'MULTILINE'` mode also matches immediately after each newline.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 109);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 112)

Unknown directive type "index".

```
.. index:: single: $ (dollar); in regular expressions
```

Matches the end of the string or just before the newline at the end of the string, and in `:const:'MULTILINE'` mode also matches before a newline. `foo` matches both 'foo' and 'foobar', while the regular expression `foo$` matches only 'foo'. More interestingly, searching for `foo.$` in 'foo1\nfoo2\n' matches 'foo2' normally, but 'foo1' in `:const:'MULTILINE'` mode; searching for a single `$` in 'foo\n' will find two (empty) matches: one just before the newline, and one at the end of the string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 115);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 115);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 123)

Unknown directive type "index".

```
.. index:: single: * (asterisk); in regular expressions
```

Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible. `ab*` will match 'a', 'ab', or 'a' followed by any number of 'b's.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 130)

Unknown directive type "index".

```
.. index:: single: + (plus); in regular expressions
```

+

Causes the resulting RE to match 1 or more repetitions of the preceding RE. `ab+` will match 'a' followed by any non-zero number of 'b's; it will not match just 'a'.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 137)

Unknown directive type "index".

```
.. index:: single: ? (question mark); in regular expressions
```

?

Causes the resulting RE to match 0 or 1 repetitions of the preceding RE. `ab?` will match either 'a' or 'ab'.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 143)

Unknown directive type "index".

```
.. index::
   single: *?; in regular expressions
   single: +?; in regular expressions
   single: ??; in regular expressions
```

*?, +?, ??

The '*', '+', and '?' quantifiers are all **dfn: greedy**; they match as much text as possible. Sometimes this behaviour isn't desired; if the RE `<.*>` is matched against `'<a> b <c>'`, it will match the entire string, and not just `'<a>'`. Adding `?` after the quantifier makes it perform the match in **dfn: non-greedy** or **dfn: minimal** fashion; as few characters as possible will be matched. Using the RE `<.*?>` will match only `'<a>'`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 149);
[backlink](#)

Unknown interpreted text role "dfn".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 149);
[backlink](#)

Unknown interpreted text role "dfn".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 149);
[backlink](#)

Unknown interpreted text role "dfn".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 157)

Unknown directive type "index".

```
.. index::
   single: *+; in regular expressions
   single: ++; in regular expressions
   single: ?+; in regular expressions
```

*+, ++, ?+

Like the '*', '+', and '?' quantifiers, those where '+' is appended also match as many times as possible. However, unlike the true greedy quantifiers, these do not allow back-tracking when the expression following it fails to match. These are known as **dfn: 'possessive'** quantifiers. For example, `a*a` will match `'aaaa'` because the `a*` will match all 4 'a's, but, when the final 'a' is encountered, the expression is backtracked so that in the end the `a*` ends up matching 3 'a's total, and the fourth 'a' is matched by the final 'a'. However, when `a*+a` is used to match `'aaaa'`, the `a*+` will match all 4 'a's, but when the final 'a' fails to find any more characters to match, the expression cannot be backtracked and will thus fail to match. `x*+`, `x++` and `x?+` are equivalent to `(?>x*)`, `(?>x+)` and `(?>x?)` correspondingly.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 163); backlink

Unknown interpreted text role "dfn".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 179)

Unknown directive type "versionadded".

```
.. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 181)

Unknown directive type "index".

```
.. index::
   single: {} (curly brackets); in regular expressions
```

`{m}`

Specifies that exactly *m* copies of the previous RE should be matched; fewer matches cause the entire RE not to match. For example, `a{6}` will match exactly six 'a' characters, but not five.

`{m,n}`

Causes the resulting RE to match from *m* to *n* repetitions of the preceding RE, attempting to match as many repetitions as possible. For example, `a{3,5}` will match from 3 to 5 'a' characters. Omitting *m* specifies a lower bound of zero, and omitting *n* specifies an infinite upper bound. As an example, `a{4,}b` will match `'aaaab'` or a thousand 'a' characters followed by a 'b', but not `'aaab'`. The comma may not be omitted or the modifier would be confused with the previously described form.

`{m,n}?`

Causes the resulting RE to match from *m* to *n* repetitions of the preceding RE, attempting to match as *few* repetitions as possible. This is the non-greedy version of the previous quantifier. For example, on the 6-character string `'aaaaaa'`, `a{3,5}` will match 5 'a' characters, while `a{3,5}?` will only match 3 characters.

`{m,n}+`

Causes the resulting RE to match from *m* to *n* repetitions of the preceding RE, attempting to match as many repetitions as possible *without* establishing any backtracking points. This is the possessive version of the quantifier above. For example, on the 6-character string `'aaaaaa'`, `a{3,5}+aa` attempt to match 5 'a' characters, then, requiring 2 more 'a's, will need more characters than available and thus fail, while `a{3,5}aa` will match with `a{3,5}` capturing 5, then 4 'a's by backtracking and then the final 2 'a's are matched by the final `aa` in the pattern. `x{m,n}+` is equivalent to `(?>x{m,n})`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 218)

Unknown directive type "versionadded".

```
.. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 220)

Unknown directive type "index".

```
.. index:: single: \ (backslash); in regular expressions
```

Either escapes special characters (permitting you to match characters like '*', '?', and so forth), or signals a special sequence; special sequences are discussed below.

If you're not using a raw string to express the pattern, remember that Python also uses the backslash as an escape sequence in string literals; if the escape sequence isn't recognized by Python's parser, the backslash and subsequent character are included in the resulting string. However, if Python would recognize the resulting sequence, the backslash should be repeated twice. This is complicated and hard to understand, so it's highly recommended that you use raw strings for all but the simplest expressions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 235)

Unknown directive type "index".

```
.. index::
   single: [] (square brackets); in regular expressions
```

[]

Used to indicate a set of characters. In a set:

- Characters can be listed individually, e.g. [amk] will match 'a', 'm', or 'k'.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 244)

Unknown directive type "index".

```
.. index:: single: - (minus); in regular expressions
```

- Ranges of characters can be indicated by giving two characters and separating them by a '-', for example [a-z] will match any lowercase ASCII letter, [0-5][0-9] will match all the two-digits numbers from 00 to 59, and [0-9A-Fa-f] will match any hexadecimal digit. If - is escaped (e.g. [a\ -z]) or if it's placed as the first or last character (e.g. [-a] or [a-]), it will match a literal '- '.
- Special characters lose their special meaning inside sets. For example, [(+*)] will match any of the literal characters '(', '+', '*', or ') '.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 257)

Unknown directive type "index".

```
.. index:: single: \ (backslash); in regular expressions
```

- Character classes such as \w or \s (defined below) are also accepted inside a set, although the characters they match depends on whether :const:`ASCII` or :const:`LOCALE` mode is in force.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 259); [backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 259); [backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 263)

Unknown directive type "index".

```
.. index:: single: ^ (caret); in regular expressions
```

- Characters that are not within a range can be matched by `:dfn:`complementing`` the set. If the first character of the set is `^`, all the characters that are *not* in the set will be matched. For example, `[^5]` will match any character except `'5'`, and `[^]` will match any character except `^`. `^` has no special meaning if it's not the first character in the set.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 265); [backlink](#)

Unknown interpreted text role "dfn".

- To match a literal `']'` inside a set, precede it with a backslash, or place it at the beginning of the set. For example, both `[() [\] {}]` and `[] () [{}]` will both match a parenthesis.
- Support of nested sets and set operations as in [Unicode Technical Standard #18](#) might be added in the future. This would change the syntax, so to facilitate this change a `:exc:`FutureWarning`` will be raised in ambiguous cases for the time being. That includes sets starting with a literal `'['` or containing literal character sequences `'--'`, `'&&'`, `'~'`, and `'|'`. To avoid a warning escape them with a backslash.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 281); [backlink](#)

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 291)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   :exc:`FutureWarning` is raised if a character set contains constructs
   that will change semantically in the future.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 295)

Unknown directive type "index".

```
.. index:: single: | (vertical bar); in regular expressions
```

`A|B`, where *A* and *B* can be arbitrary REs, creates a regular expression that will match either *A* or *B*. An arbitrary number of REs can be separated by the `'|'` in this way. This can be used inside groups (see below) as well. As the target string is scanned, REs separated by `'|'` are tried from left to right. When one pattern completely matches, that branch is accepted. This means that once *A* matches, *B* will not be tested further, even if it would produce a longer overall match. In other words, the `'|'` operator is never greedy. To match a literal `'|'`, use `\|`, or enclose it inside a character class, as in `[|]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 308)

Unknown directive type "index".

```
.. index::
   single: () (parentheses); in regular expressions
```

(...)

Matches whatever regular expression is inside the parentheses, and indicates the start and end of a group; the contents of a group can be retrieved after a match has been performed, and can be matched later in the string with the `\number` special sequence, described below. To match the literals `'('` or `')'`, use `\(` or `\)`, or enclose them inside a character class: `[()]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 318)

Unknown directive type "index".

```
.. index:: single: (?; in regular expressions
```

(?...)

This is an extension notation (a '?' following a '(' is not meaningful otherwise). The first character after the '?' determines what the meaning and further syntax of the construct is. Extensions usually do not create a new group; (?P<name>...) is the only exception to this rule. Following are the currently supported extensions.

(?aiLmsux)

(One or more letters from the set 'a', 'i', 'L', 'm', 's', 'u', 'x'.) The group matches the empty string; the letters set the corresponding flags: `:const:'re.A'` (ASCII-only matching), `:const:'re.I'` (ignore case), `:const:'re.L'` (locale dependent), `:const:'re.M'` (multi-line), `:const:'re.S'` (dot matches all), `:const:'re.U'` (Unicode matching), and `:const:'re.X'` (verbose), for the entire regular expression. (The flags are described in [ref: contents-of-module-re](#).) This is useful if you wish to include the flags as part of the regular expression, instead of passing a *flag* argument to the `:func:'re.compile'` function. Flags should be used first in the expression string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 328);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 328);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 328);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 328);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 328);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 328);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 328);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 328);
[backlink](#)

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 328);
[backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 341)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.11
    This construction can only be used at the start of the expression.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 344)

Unknown directive type "index".

```
.. index:: single: (?; in regular expressions
```

(?:...)

A non-capturing version of regular parentheses. Matches whatever regular expression is inside the parentheses, but the substring matched by the group *cannot* be retrieved after performing a match or referenced later in the pattern.

(?aiLmsux-imsx:...)

(Zero or more letters from the set 'a', 'i', 'L', 'm', 's', 'u', 'x', optionally followed by '-' followed by one or more letters from the 'i', 'm', 's', 'x'.) The letters set or remove the corresponding flags: `:const:re.A` (ASCII-only matching), `:const:re.I` (ignore case), `:const:re.L` (locale dependent), `:const:re.M` (multi-line), `:const:re.S` (dot matches all), `:const:re.U` (Unicode matching), and `:const:re.X` (verbose), for the part of the expression. (The flags are described in [ref: contents-of-module-re](#).)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 353);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 353);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 353);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 353);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 353);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 353);

[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 353);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 353);
[backlink](#)

Unknown interpreted text role "ref".

The letters 'a', 'L' and 'u' are mutually exclusive when used as inline flags, so they can't be combined or follow '-'. Instead, when one of them appears in an inline group, it overrides the matching mode in the enclosing group. In Unicode patterns (?a:...) switches to ASCII-only matching, and (?u:...) switches to Unicode matching (default). In byte pattern (?L:...) switches to locale depending matching, and (?a:...) switches to ASCII-only matching (default). This override is only in effect for the narrow inline group, and the original matching mode is restored outside of the group.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 373)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 375)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   The letters ``'a'``, ``'L'`` and ``'u'`` also can be used in a group.
```

(?>...)

Attempts to match ... as if it was a separate regular expression, and if successful, continues to match the rest of the pattern following it. If the subsequent pattern fails to match, the stack can only be unwound to a point *before* the (?>...) because once exited, the expression, known as an **dfn:'atomic group'**, has thrown away all stack points within itself. Thus, (?>.*). would never match anything because first the .* would match all characters possible, then, having nothing left to match, the final . would fail to match. Since there are no stack points saved in the Atomic Group, and there is no stack point before it, the entire expression would thus fail to match.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 379);
[backlink](#)

Unknown interpreted text role "dfn".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 391)

Unknown directive type "versionadded".

```
.. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 393)

Unknown directive type "index".

```
.. index:: single: (?P<; in regular expressions
```

(?P<name>...)

Similar to regular parentheses, but the substring matched by the group is accessible via the symbolic group name *name*. Group names must be valid Python identifiers, and each group name must be defined only once within a regular expression. A symbolic group is also a numbered group, just as if the group were not named.

Named groups can be referenced in three contexts. If the pattern is `(?P<quote>['"])*?(?P=quote)` (i.e. matching a string quoted with either single or double quotes):

Context of reference to group "quote"	Ways to reference it
in the same pattern itself	<ul style="list-style-type: none">• <code>(?P=quote)</code> (as shown)• <code>\1</code>
when processing match object <i>m</i>	<ul style="list-style-type: none">• <code>m.group('quote')</code>• <code>m.end('quote')</code> (etc.)
in a string passed to the <i>repl</i> argument of <code>re.sub()</code>	<ul style="list-style-type: none">• <code>\g<quote></code>• <code>\g<1></code>• <code>\1</code>

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 420)

Unknown directive type "index".

```
.. index:: single: (?P=; in regular expressions
```

(?P=name)

A backreference to a named group; it matches whatever text was matched by the earlier group named *name*.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 426)

Unknown directive type "index".

```
.. index:: single: (?#; in regular expressions
```

(?#...)

A comment; the contents of the parentheses are simply ignored.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 431)

Unknown directive type "index".

```
.. index:: single: (?=; in regular expressions
```

(?=...)

Matches if ... matches next, but doesn't consume any of the string. This is called a `'dfn': lookahead assertion`. For example, `Isaac (?=Asimov)` will match `'Isaac '` only if it's followed by `'Asimov'`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 434);
[backlink](#)

Unknown interpreted text role "dfn".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 438)

Unknown directive type "index".

```
.. index:: single: (?!; in regular expressions
```

(?!...)

Matches if ... doesn't match next. This is a `:dfn:'negative lookahead assertion'`. For example, `Isaac (?!Asimov)` will match `'Isaac '` only if it's *not* followed by `'Asimov'`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 441);
[backlink](#)

Unknown interpreted text role "dfn".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 445)

Unknown directive type "index".

```
.. index:: single: (?<=; in regular expressions
```

(?<=...)

Matches if the current position in the string is preceded by a match for ... that ends at the current position. This is called a `:dfn:'positive lookbehind assertion'`. `(?<=abc) def` will find a match in `'abcdef'`, since the lookbehind will back up 3 characters and check if the contained pattern matches. The contained pattern must only match strings of some fixed length, meaning that `abc` or `a|b` are allowed, but `a*` and `a{3,4}` are not. Note that patterns which start with positive lookbehind assertions will not match at the beginning of the string being searched; you will most likely want to use the `:func:'search'` function rather than the `:func:'match'` function:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 448);
[backlink](#)

Unknown interpreted text role "dfn".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 448);
[backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 448);
[backlink](#)

Unknown interpreted text role "func".

```
>>> import re
>>> m = re.search('(?<=abc) def', 'abcdef')
>>> m.group(0)
'def'
```

This example looks for a word following a hyphen:

```
>>> m = re.search(r'(?<=-)\w+', 'spam-egg')
>>> m.group(0)
'egg'
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 469)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.5
   Added support for group references of fixed length.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 472)

Unknown directive type "index".

```
.. index:: single: (?<!; in regular expressions
```

(?<!...)

Matches if the current position in the string is not preceded by a match for ... This is called a **dfn:'negative lookbehind assertion'**. Similar to positive lookbehind assertions, the contained pattern must only match strings of some fixed length. Patterns which start with negative lookbehind assertions may match at the beginning of the string being searched.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 475); backlink

Unknown interpreted text role "dfn".

(? (id/name) yes-pattern|no-pattern)

Will try to match with `yes-pattern` if the group with given *id* or *name* exists, and with `no-pattern` if it doesn't. `no-pattern` is optional and can be omitted. For example, `(<) ? (\w+@ \w+ (? : \. \w+) +) (? (1) > | $)` is a poor email matching pattern, which will match with `'<user@host.com>'` as well as `'user@host.com'`, but not with `'<user@host.com'` nor `'user@host.com>'`.

The special sequences consist of `'\'` and a character from the list below. If the ordinary character is not an ASCII digit or an ASCII letter, then the resulting RE will match the second character. For example, `\$` matches the character `'$'`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 495)

Unknown directive type "index".

.. index:: single: \ (backslash); in regular expressions

`\number`

Matches the contents of the group of the same number. Groups are numbered starting from 1. For example, `(.+) \1` matches `'the the'` or `'55 55'`, but not `'thethe'` (note the space after the group). This special sequence can only be used to match one of the first 99 groups. If the first digit of *number* is 0, or *number* is 3 octal digits long, it will not be interpreted as a group match, but as the character with octal value *number*. Inside the `'['` and `']'` of a character class, all numeric escapes are treated as characters.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 507)

Unknown directive type "index".

.. index:: single: \A; in regular expressions

`\A`

Matches only at the start of the string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 512)

Unknown directive type "index".

.. index:: single: \b; in regular expressions

`\b`

Matches the empty string, but only at the beginning or end of a word. A word is defined as a sequence of word characters. Note that formally, `\b` is defined as the boundary between a `\w` and a `\W` character (or vice versa), or between `\w` and the beginning/end of the string. This means that `r'\bfoo\b'` matches `'foo'`, `'foo.'`, `'(foo)'`, `'bar foo baz'` but not `'foobar'` or `'foo3'`.

By default Unicode alphanumerics are the ones used in Unicode patterns, but this can be changed by using the **const:'ASCII'** flag. Word boundaries are determined by the current locale if the **const:'LOCALE'** flag is used. Inside a character range, `\b` represents the backspace character, for compatibility with Python's string literals.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 522);

[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 522);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 528)

Unknown directive type "index".

```
.. index:: single: \B; in regular expressions
```

\B

Matches the empty string, but only when it is *not* at the beginning or end of a word. This means that `r'py\B'` matches `'python'`, `'py3'`, `'py2'`, but not `'py'`, `'py.'`, or `'py!'`. `\B` is just the opposite of `\b`, so word characters in Unicode patterns are Unicode alphanumerics or the underscore, although this can be changed by using the `:const:'ASCII'` flag. Word boundaries are determined by the current locale if the `:const:'LOCALE'` flag is used.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 531);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 531);
[backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 539)

Unknown directive type "index".

```
.. index:: single: \d; in regular expressions
```

\d

For Unicode (str) patterns:

Matches any Unicode decimal digit (that is, any character in Unicode character category `[Nd]`). This includes `[0-9]`, and also many other digit characters. If the `:const:'ASCII'` flag is used only `[0-9]` is matched.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 543);
[backlink](#)

Unknown interpreted text role "const".

For 8-bit (bytes) patterns:

Matches any decimal digit; this is equivalent to `[0-9]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 551)

Unknown directive type "index".

```
.. index:: single: \D; in regular expressions
```

`\D`

Matches any character which is not a decimal digit. This is the opposite of `\d`. If the `:const:'ASCII'` flag is used this becomes the equivalent of `[\^0-9]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 554); [backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 558)

Unknown directive type "index".

```
.. index:: single: \s; in regular expressions
```

`\s`

For Unicode (str) patterns:

Matches Unicode whitespace characters (which includes `[\t\n\r\f\v]`, and also many other characters, for example the non-breaking spaces mandated by typography rules in many languages). If the `:const:'ASCII'` flag is used, only `[\t\n\r\f\v]` is matched.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 562); [backlink](#)

Unknown interpreted text role "const".

For 8-bit (bytes) patterns:

Matches characters considered whitespace in the ASCII character set; this is equivalent to `[\t\n\r\f\v]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 572)

Unknown directive type "index".

```
.. index:: single: \S; in regular expressions
```

`\S`

Matches any character which is not a whitespace character. This is the opposite of `\s`. If the `:const:'ASCII'` flag is used this becomes the equivalent of `[\^ \t\n\r\f\v]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 575); [backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 579)

Unknown directive type "index".

```
.. index:: single: \w; in regular expressions
```

`\w`

For Unicode (str) patterns:

Matches Unicode word characters; this includes most characters that can be part of a word in any language, as well as numbers and the underscore. If the `:const:'ASCII'` flag is used, only `[a-zA-Z0-9_]` is matched.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-

resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 583); [backlink](#)

Unknown interpreted text role "const".

For 8-bit (bytes) patterns:

Matches characters considered alphanumeric in the ASCII character set; this is equivalent to `[a-zA-Z0-9_]`. If the `:const:'LOCALE'` flag is used, matches characters considered alphanumeric in the current locale and the underscore.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 589); [backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 594)

Unknown directive type "index".

```
.. index:: single: \W; in regular expressions
```

\W

Matches any character which is not a word character. This is the opposite of `\w`. If the `:const:'ASCII'` flag is used this becomes the equivalent of `[^a-zA-Z0-9_]`. If the `:const:'LOCALE'` flag is used, matches characters which are neither alphanumeric in the current locale nor the underscore.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 597); [backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 597); [backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 603)

Unknown directive type "index".

```
.. index:: single: \Z; in regular expressions
```

\Z

Matches only at the end of the string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 608)

Unknown directive type "index".

```
.. index::
  single: \a; in regular expressions
  single: \b; in regular expressions
  single: \f; in regular expressions
  single: \n; in regular expressions
  single: \N; in regular expressions
  single: \r; in regular expressions
  single: \t; in regular expressions
  single: \u; in regular expressions
  single: \U; in regular expressions
  single: \v; in regular expressions
  single: \x; in regular expressions
```

```
single: \\; in regular expressions
```

Most of the standard escapes supported by Python string literals are also accepted by the regular expression parser:

<code>\a</code>	<code>\b</code>	<code>\f</code>	<code>\n</code>
<code>\N</code>	<code>\r</code>	<code>\t</code>	<code>\u</code>
<code>\U</code>	<code>\v</code>	<code>\x</code>	<code>\\</code>

(Note that `\b` is used to represent word boundaries, and means "backspace" only inside character classes.)

'`\u`', '`\U`', and '`\N`' escape sequences are only recognized in Unicode patterns. In bytes patterns they are errors. Unknown escapes of ASCII letters are reserved for future use and treated as errors.

Octal escapes are included in a limited form. If the first digit is a 0, or if there are three octal digits, it is considered an octal escape. Otherwise, it is a group reference. As for string literals, octal escapes are always at most three digits in length.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 641)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   The ``\u`` and ``\U`` escape sequences have been added.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 644)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.6
   Unknown escapes consisting of ``\`` and an ASCII letter now are errors.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 647)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.8
   The ``\N{name}`` escape sequence has been added. As in string literals,
   it expands to the named Unicode character (e.g. ``\N{EM DASH}`).
```

Module Contents

The module defines several functions, constants, and an exception. Some of the functions are simplified versions of the full featured methods for compiled regular expressions. Most non-trivial applications always use the compiled form.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 662)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.6
   Flag constants are now instances of :class:`RegexFlag`, which is a subclass of
   :class:`enum.IntFlag`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 666)

Unknown directive type "function".

```
.. function:: compile(pattern, flags=0)
```

Compile a regular expression pattern into a :ref:`regular expression object <re-objects>`, which can be used for matching using its :func:`~Pattern.match`, :func:`~Pattern.search` and other methods, described below.

The expression's behaviour can be modified by specifying a `*flags*` value. Values can be any of the following variables, combined using bitwise OR (the

```

    ``|`` operator).

The sequence ::

    prog = re.compile(pattern)
    result = prog.match(string)

is equivalent to ::

    result = re.match(pattern, string)

but using :func:`re.compile` and saving the resulting regular expression
object for reuse is more efficient when the expression will be used several
times in a single program.

.. note::

    The compiled versions of the most recent patterns passed to
    :func:`re.compile` and the module-level matching functions are cached, so
    programs that use only a few regular expressions at a time needn't worry
    about compiling regular expressions.

```

An `:class:`enum.IntFlag`` class containing the regex options listed below.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 699); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 701)

Unknown directive type "versionadded".

```
.. versionadded:: 3.11 - added to ``__all__``
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 703)

Unknown directive type "data".

```
.. data:: A
    ASCII

    Make ``\w``, ``\W``, ``\b``, ``\B``, ``\d``, ``\D``, ``\s`` and ``\S``
    perform ASCII-only matching instead of full Unicode matching. This is only
    meaningful for Unicode patterns, and is ignored for byte patterns.
    Corresponds to the inline flag ``(?a)``.

    Note that for backward compatibility, the :const:`re.U` flag still
    exists (as well as its synonym :const:`re.UNICODE` and its embedded
    counterpart ``(?u)``), but these are redundant in Python 3 since
    matches are Unicode by default for strings (and Unicode matching
    isn't allowed for bytes).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 718)

Unknown directive type "data".

```
.. data:: DEBUG

    Display debug information about compiled expression.
    No corresponding inline flag.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 724)

Unknown directive type "data".

```
.. data:: I
    IGNORECASE
```

Perform case-insensitive matching; expressions like ```[A-Z]``` will also match lowercase letters. Full Unicode matching (such as ```Ãæ``` matching ```Ã```) also works unless the `:const:re.ASCII`` flag is used to disable non-ASCII matches. The current locale does not change the effect of this flag unless the `:const:re.LOCALE`` flag is also used. Corresponds to the inline flag ```(?i)```.

Note that when the Unicode patterns ```[a-z]``` or ```[A-Z]``` are used in combination with the `:const:IGNORECASE`` flag, they will match the 52 ASCII letters and 4 additional non-ASCII letters: 'Ã' (U+0130, Latin capital letter I with dot above), 'Ã±' (U+0131, Latin small letter dotless i), 'Ã¸' (U+017F, Latin small letter long s) and 'Ã¸' (U+212A, Kelvin sign). If the `:const:ASCII`` flag is used, only letters 'a' to 'z' and 'A' to 'Z' are matched.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library)re.rst, line 742)

Unknown directive type "data".

```
.. data:: L
    LOCALE
```

Make ```w```, ```W```, ```b```, ```B``` and case-insensitive matching dependent on the current locale. This flag can be used only with bytes patterns. The use of this flag is discouraged as the locale mechanism is very unreliable, it only handles one "culture" at a time, and it only works with 8-bit locales. Unicode matching is already enabled by default in Python 3 for Unicode (str) patterns, and it is able to handle different locales/languages. Corresponds to the inline flag ```(?L)```.

```
.. versionchanged:: 3.6
    :const:re.LOCALE` can be used only with bytes patterns and is
    not compatible with :const:re.ASCII`.
```

```
.. versionchanged:: 3.7
    Compiled regular expression objects with the :const:re.LOCALE` flag no
    longer depend on the locale at compile time. Only the locale at
    matching time affects the result of matching.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library)re.rst, line 764)

Unknown directive type "data".

```
.. data:: M
    MULTILINE
```

When specified, the pattern character ```^``` matches at the beginning of the string and at the beginning of each line (immediately following each newline); and the pattern character ```$``` matches at the end of the string and at the end of each line (immediately preceding each newline). By default, ```^``` matches only at the beginning of the string, and ```$``` only at the end of the string and immediately before the newline (if any) at the end of the string. Corresponds to the inline flag ```(?m)```.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library)re.rst, line 775)

Unknown directive type "data".

```
.. data:: NOFLAG
```

Indicates no flag being applied, the value is ```0```. This flag may be used as a default value for a function keyword argument or as a base value that will be conditionally ORed with other flags. Example of use as a default value::

```
def myfunc(text, flag=re.NOFLAG):
    return re.match(text, flag)
```

```
.. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 787)

Unknown directive type "data".

```
.. data:: S
    DOTALL
```

Make the ```'.``` special character match any character at all, including a newline; without this flag, ```'.``` will match anything *except* a newline. Corresponds to the inline flag ```(?s)```.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 795)

Unknown directive type "data".

```
.. data:: X
    VERBOSE
```

```
.. index:: single: # (hash); in regular expressions
```

This flag allows you to write regular expressions that look nicer and are more readable by allowing you to visually separate logical sections of the pattern and add comments. Whitespace within the pattern is ignored, except when in a character class, or when preceded by an unescaped backslash, or within tokens like ```*?```, ```(?:``` or ```(?P<...>```. When a line contains a ```#``` that is not in a character class and is not preceded by an unescaped backslash, all characters from the leftmost such ```#``` through the end of the line are ignored.

This means that the two following regular expression objects that match a decimal number are functionally equal::

```
a = re.compile(r"""\d + # the integral part
                \.  # the decimal point
                \d * # some fractional digits""", re.X)
b = re.compile(r"\d+\.\d*")
```

Corresponds to the inline flag ```(?x)```.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 820)

Unknown directive type "function".

```
.. function:: search(pattern, string, flags=0)
```

Scan through **string** looking for the first location where the regular expression **pattern** produces a match, and return a corresponding :ref:`match object <match-objects>`. Return ```None``` if no position in the string matches the pattern; note that this is different from finding a zero-length match at some point in the string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 829)

Unknown directive type "function".

```
.. function:: match(pattern, string, flags=0)
```

If zero or more characters at the beginning of **string** match the regular expression **pattern**, return a corresponding :ref:`match object <match-objects>`. Return ```None``` if the string does not match the pattern; note that this is different from a zero-length match.

Note that even in :const:`MULTILINE` mode, :func:`re.match` will only match

at the beginning of the string and not at the beginning of each line.

If you want to locate a match anywhere in `*string*`, use `:func:`search`` instead (see also `:ref:`search-vs-match``).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library)re.rst, line 843)

Unknown directive type "function".

```
.. function:: fullmatch(pattern, string, flags=0)
```

If the whole `*string*` matches the regular expression `*pattern*`, return a corresponding `:ref:`match object <match-objects>``. Return ``None`` if the string does not match the pattern; note that this is different from a zero-length match.

```
.. versionadded:: 3.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library)re.rst, line 853)

Unknown directive type "function".

```
.. function:: split(pattern, string, maxsplit=0, flags=0)
```

Split `*string*` by the occurrences of `*pattern*`. If capturing parentheses are used in `*pattern*`, then the text of all groups in the pattern are also returned as part of the resulting list. If `*maxsplit*` is nonzero, at most `*maxsplit*` splits occur, and the remainder of the string is returned as the final element of the list. ::

```
>>> re.split(r'\W+', 'Words, words, words.')
['Words', 'words', 'words', '']
>>> re.split(r'(\W+)', 'Words, words, words.')
['Words', ',', ' ', 'words', ',', ' ', 'words', ' ', '']
>>> re.split(r'\W+', 'Words, words, words.', 1)
['Words', 'words, words.']
>>> re.split('[a-f]+', '0a3B9', flags=re.IGNORECASE)
['0', '3', '9']
```

If there are capturing groups in the separator and it matches at the start of the string, the result will start with an empty string. The same holds for the end of the string::

```
>>> re.split(r'(\W+)', '...words, words...')
['', '...', 'words', ',', ' ', 'words', '...', '']
```

That way, separator components are always found at the same relative indices within the result list.

Empty matches for the pattern split the string only when not adjacent to a previous empty match.

```
>>> re.split(r'\b', 'Words, words, words.')
['', 'Words', ',', ' ', 'words', ',', ' ', 'words', '']
>>> re.split(r'\W*', '...words...')
['', ' ', 'w', 'o', 'r', 'd', 's', ' ', '']
>>> re.split(r'(\W*)', '...words...')
['', '...', ' ', ' ', 'w', ' ', 'o', ' ', 'r', ' ', 'd', ' ', 's', ' ', '...', ' ', ' ', '']
```

```
.. versionchanged:: 3.1
   Added the optional flags argument.
```

```
.. versionchanged:: 3.7
   Added support of splitting on a pattern that could match an empty string.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library)re.rst, line 897)

Unknown directive type "function".

```
.. function:: findall(pattern, string, flags=0)
```

Return all non-overlapping matches of **pattern** in **string**, as a list of strings or tuples. The **string** is scanned left-to-right, and matches are returned in the order found. Empty matches are included in the result.

The result depends on the number of capturing groups in the pattern. If there are no groups, return a list of strings matching the whole pattern. If there is exactly one group, return a list of strings matching that group. If multiple groups are present, return a list of tuples of strings matching the groups. Non-capturing groups do not affect the form of the result.

```
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.findall(r'(\w+)=(\d+)', 'set width=20 and height=10')
[('width', '20'), ('height', '10')]
```

```
.. versionchanged:: 3.7
```

Non-empty matches can now start just after a previous empty match.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 919)

Unknown directive type "function".

```
.. function:: finditer(pattern, string, flags=0)
```

Return an `:term:`iterator`` yielding `:ref:`match objects <match-objects>`` over all non-overlapping matches for the RE **pattern** in **string**. The **string** is scanned left-to-right, and matches are returned in the order found. Empty matches are included in the result.

```
.. versionchanged:: 3.7
```

Non-empty matches can now start just after a previous empty match.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 930)

Unknown directive type "function".

```
.. function:: sub(pattern, repl, string, count=0, flags=0)
```

Return the string obtained by replacing the leftmost non-overlapping occurrences of **pattern** in **string** by the replacement **repl**. If the pattern isn't found, **string** is returned unchanged. **repl** can be a string or a function; if it is a string, any backslash escapes in it are processed. That is, ```\n``` is converted to a single newline character, ```\r``` is converted to a carriage return, and so forth. Unknown escapes of ASCII letters are reserved for future use and treated as errors. Other unknown escapes such as ```&``` are left alone. Backreferences, such as ```\6```, are replaced with the substring matched by group 6 in the pattern. For example::

```
>>> re.sub(r'def\s+([a-zA-Z_][a-zA-Z_0-9]*)\s*(\s*):',
...       r'static PyObject*\np_\1(void)\n{',
...       'def myfunc():')
'static PyObject*\np_myfunc(void)\n{'
```

If **repl** is a function, it is called for every non-overlapping occurrence of **pattern**. The function takes a single `:ref:`match object <match-objects>`` argument, and returns the replacement string. For example::

```
>>> def dashrepl(matchobj):
...     if matchobj.group(0) == '-': return ' '
...     else: return '-'
>>> re.sub('-{1,2}', dashrepl, 'pro---gram-files')
'pro--gram files'
>>> re.sub(r'\sAND\s', ' & ', 'Baked Beans And Spam', flags=re.IGNORECASE)
'Baked Beans & Spam'
```

The pattern may be a string or a `:ref:`pattern object <re-objects>``.

The optional argument **count** is the maximum number of pattern occurrences to be replaced; **count** must be a non-negative integer. If omitted or zero, all

occurrences will be replaced. Empty matches for the pattern are replaced only when not adjacent to a previous empty match, so `sub('x*', '-', 'abxd')` returns `'-a-b--d-'`.

`.. index:: single: \g; in regular expressions`

In string-type `*repl*` arguments, in addition to the character escapes and backreferences described above, `g<name>` will use the substring matched by the group named `name`, as defined by the `(?P<name>...)` syntax. `g<number>` uses the corresponding group number; `g<2>` is therefore equivalent to `2`, but isn't ambiguous in a replacement such as `g<2>0`. `20` would be interpreted as a reference to group 20, not a reference to group 2 followed by the literal character `'0'`. The backreference `g<0>` substitutes in the entire substring matched by the RE.

`.. versionchanged:: 3.1`
Added the optional flags argument.

`.. versionchanged:: 3.5`
Unmatched groups are replaced with an empty string.

`.. versionchanged:: 3.6`
Unknown escapes in `*pattern*` consisting of `'\'` and an ASCII letter now are errors.

`.. versionchanged:: 3.7`
Unknown escapes in `*repl*` consisting of `'\'` and an ASCII letter now are errors.

`.. versionchanged:: 3.7`
Empty matches for the pattern are replaced when adjacent to a previous non-empty match.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 999)

Unknown directive type "function".

`.. function:: subn(pattern, repl, string, count=0, flags=0)`

Perform the same operation as `:func:sub`, but return a tuple `(new_string, number_of_subs_made)`.

`.. versionchanged:: 3.1`
Added the optional flags argument.

`.. versionchanged:: 3.5`
Unmatched groups are replaced with an empty string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1011)

Unknown directive type "function".

`.. function:: escape(pattern)`

Escape special characters in `*pattern*`. This is useful if you want to match an arbitrary literal string that may have regular expression metacharacters in it. For example::

```
>>> print(re.escape('https://www.python.org'))
https://www\.python\.org

>>> legal_chars = string.ascii_lowercase + string.digits + " !#$%&'*+-.^_`|~:"
>>> print('[%s]+' % re.escape(legal_chars))
[abcdefghijklmnopqrstuvwxyz0123456789!\#$%&'*\+,-.\^_`|\~:]+

>>> operators = ['+', '-', '*', '/', '**']
>>> print('|'.join(map(re.escape, sorted(operators, reverse=True))))
/|-|\+|\*|\*|
```

This function must not be used for the replacement string in `:func:sub` and `:func:subn`, only backslashes should be escaped. For example::

```
>>> digits_re = r'\d+'
>>> sample = '/usr/sbin/sendmail - 0 errors, 12 warnings'
>>> print(re.sub(digits_re, digits_re.replace('\d', r'\d'), sample))
/usr/sbin/sendmail - \d+ errors, \d+ warnings

.. versionchanged:: 3.3
    The ``_`` character is no longer escaped.

.. versionchanged:: 3.7
    Only characters that can have special meaning in a regular expression
    are escaped. As a result, ``!``, ``*``, ``%``, ``&``, ``'``, ``(``,
    ``/``, ``:``, ``;``, ``<``, ``=``, ``>``, ``@``, and
    ``"`` are no longer escaped.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1046)

Unknown directive type "function".

```
.. function:: purge()

    Clear the regular expression cache.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1051)

Unknown directive type "exception".

```
.. exception:: error(msg, pattern=None, pos=None)

    Exception raised when a string passed to one of the functions here is not a
    valid regular expression (for example, it might contain unmatched parentheses)
    or when some other error occurs during compilation or matching. It is never an
    error if a string contains no match for a pattern. The error instance has
    the following additional attributes:

    .. attribute:: msg

        The unformatted error message.

    .. attribute:: pattern

        The regular expression pattern.

    .. attribute:: pos

        The index in *pattern* where compilation failed (may be ``None``).

    .. attribute:: lineno

        The line corresponding to *pos* (may be ``None``).

    .. attribute:: colno

        The column corresponding to *pos* (may be ``None``).

.. versionchanged:: 3.5
    Added additional attributes.
```

Regular Expression Objects

Compiled regular expression objects support the following methods and attributes:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1090)

Unknown directive type "method".

```
.. method:: Pattern.search(string[, pos[, endpos]])

    Scan through *string* looking for the first location where this regular
    expression produces a match, and return a corresponding :ref:`match object`
```

<match-objects>`. Return ``None`` if no position in the string matches the pattern; note that this is different from finding a zero-length match at some point in the string.

The optional second parameter **pos** gives an index in the string where the search is to start; it defaults to ``0``. This is not completely equivalent to slicing the string; the ``'^'`` pattern character matches at the real beginning of the string and at positions just after a newline, but not necessarily at the index where the search is to start.

The optional parameter **endpos** limits how far the string will be searched; it will be as if the string is **endpos** characters long, so only the characters from **pos** to **endpos* - 1* will be searched for a match. If **endpos** is less than **pos**, no match will be found; otherwise, if **rx** is a compiled regular expression object, `rx.search(string, 0, 50)` is equivalent to `rx.search(string[:50], 0)`.

```
>>> pattern = re.compile("d")
>>> pattern.search("dog")      # Match at index 0
<re.Match object; span=(0, 1), match='d'>
>>> pattern.search("dog", 1)   # No match; search doesn't include the "d"
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library)re.rst, line 1117)

Unknown directive type "method".

```
.. method:: Pattern.match(string[, pos[, endpos]])
```

If zero or more characters at the **beginning** of **string** match this regular expression, return a corresponding :ref:`match object` <match-objects>`. Return ``None`` if the string does not match the pattern; note that this is different from a zero-length match.

The optional **pos** and **endpos** parameters have the same meaning as for the :meth:`~Pattern.search` method. ::

```
>>> pattern = re.compile("o")
>>> pattern.match("dog")      # No match as "o" is not at the start of "dog".
>>> pattern.match("dog", 1)   # Match as "o" is the 2nd character of "dog".
<re.Match object; span=(1, 2), match='o'>
```

If you want to locate a match anywhere in **string**, use :meth:`~Pattern.search` instead (see also :ref:`search-vs-match`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library)re.rst, line 1136)

Unknown directive type "method".

```
.. method:: Pattern.fullmatch(string[, pos[, endpos]])
```

If the whole **string** matches this regular expression, return a corresponding :ref:`match object` <match-objects>`. Return ``None`` if the string does not match the pattern; note that this is different from a zero-length match.

The optional **pos** and **endpos** parameters have the same meaning as for the :meth:`~Pattern.search` method. ::

```
>>> pattern = re.compile("o[gh]")
>>> pattern.fullmatch("dog")   # No match as "o" is not at the start of "dog".
>>> pattern.fullmatch("ogre")  # No match as not the full string matches.
>>> pattern.fullmatch("doggie", 1, 3) # Matches within given limits.
<re.Match object; span=(1, 3), match='og'>
```

```
.. versionadded:: 3.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library)re.rst, line 1154)

Unknown directive type "method".

```
.. method:: Pattern.split(string, maxsplit=0)
```

Identical to the :func:`split` function, using the compiled pattern.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1159)

Unknown directive type "method".

```
.. method:: Pattern.findall(string[, pos[, endpos]])
```

Similar to the :func:`findall` function, using the compiled pattern, but also accepts optional *pos* and *endpos* parameters that limit the search region like for :meth:`search`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1166)

Unknown directive type "method".

```
.. method:: Pattern.finditer(string[, pos[, endpos]])
```

Similar to the :func:`finditer` function, using the compiled pattern, but also accepts optional *pos* and *endpos* parameters that limit the search region like for :meth:`search`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1173)

Unknown directive type "method".

```
.. method:: Pattern.sub(repl, string, count=0)
```

Identical to the :func:`sub` function, using the compiled pattern.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1178)

Unknown directive type "method".

```
.. method:: Pattern.subn(repl, string, count=0)
```

Identical to the :func:`subn` function, using the compiled pattern.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1183)

Unknown directive type "attribute".

```
.. attribute:: Pattern.flags
```

The regex matching flags. This is a combination of the flags given to :func:`.compile`, any ``(?...)`` inline flags in the pattern, and implicit flags such as :data:`UNICODE` if the pattern is a Unicode string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1190)

Unknown directive type "attribute".

```
.. attribute:: Pattern.groups
```

The number of capturing groups in the pattern.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1195)

Unknown directive type "attribute".

```
.. attribute:: Pattern.groupindex
```

A dictionary mapping any symbolic group names defined by ``(?P<id>`` to group numbers. The dictionary is empty if no symbolic groups were used in the pattern.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1202)

Unknown directive type "attribute".

```
.. attribute:: Pattern.pattern
```

The pattern string from which the pattern object was compiled.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1207)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   Added support of :func:`copy.copy` and :func:`copy.deepcopy`. Compiled
   regular expression objects are considered atomic.
```

Match Objects

Match objects always have a boolean value of `True`. Since `:meth:`~Pattern.match`` and `:meth:`~Pattern.search`` return `None` when there is no match, you can test whether there was a match with a simple `if` statement:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1217); [backlink](#)

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1217); [backlink](#)

Unknown interpreted text role "meth".

```
match = re.search(pattern, string)
if match:
    process(match)
```

Match objects support the following methods and attributes:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1229)

Unknown directive type "method".

```
.. method:: Match.expand(template)
```

Return the string obtained by doing backslash substitution on the template string `*template*`, as done by the `:meth:`~Pattern.sub`` method. Escapes such as ```\n``` are converted to the appropriate characters, and numeric backreferences (```\1```, ```\2```) and named backreferences (```\g<1>```, ```\g<name>```) are replaced by the contents of the corresponding group.

```
.. versionchanged:: 3.5
    Unmatched groups are replaced with an empty string.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1241)

Unknown directive type "method".

```
.. method:: Match.group([group1, ...])
```

Returns one or more subgroups of the match. If there is a single argument, the result is a single string; if there are multiple arguments, the result is a tuple with one item per argument. Without arguments, `*group1*` defaults to zero (the whole match is returned). If a `*groupN*` argument is zero, the corresponding return value is the entire matching string; if it is in the inclusive range `[1..99]`, it is the string matching the corresponding parenthesized group. If a group number is negative or larger than the number of groups defined in the pattern, an `:exc:`IndexError`` exception is raised. If a group is contained in a part of the pattern that did not match, the corresponding result is ```None```. If a group is contained in a part of the pattern that matched multiple times, the last match is returned. ::

```
>>> m = re.match(r"(\w+) (\w+)", "Isaac Newton, physicist")
>>> m.group(0)          # The entire match
'Isaac Newton'
>>> m.group(1)          # The first parenthesized subgroup.
'Isaac'
>>> m.group(2)          # The second parenthesized subgroup.
'Newton'
>>> m.group(1, 2)       # Multiple arguments give us a tuple.
('Isaac', 'Newton')
```

If the regular expression uses the ```(?P<name>...)``` syntax, the `*groupN*` arguments may also be strings identifying groups by their group name. If a string argument is not used as a group name in the pattern, an `:exc:`IndexError`` exception is raised.

A moderately complicated example::

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
>>> m.group('first_name')
'Malcolm'
>>> m.group('last_name')
'Reynolds'
```

Named groups can also be referred to by their index::

```
>>> m.group(1)
'Malcolm'
>>> m.group(2)
'Reynolds'
```

If a group matches multiple times, only the last match is accessible::

```
>>> m = re.match(r"(..)+", "alb2c3") # Matches 3 times.
>>> m.group(1)                       # Returns only the last match.
'c3'
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1292)

Unknown directive type "method".

```
.. method:: Match.__getitem__(g)
```

This is identical to ```m.group(g)```. This allows easier access to an individual group from a match::

```
>>> m = re.match(r"(\w+) (\w+)", "Isaac Newton, physicist")
>>> m[0]          # The entire match
'Isaac Newton'
>>> m[1]          # The first parenthesized subgroup.
'Isaac'
>>> m[2]          # The second parenthesized subgroup.
'Newton'
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1308)

Unknown directive type "method".

```
.. method:: Match.groups(default=None)
```

Return a tuple containing all the subgroups of the match, from 1 up to however many groups are in the pattern. The **default** argument is used for groups that did not participate in the match; it defaults to ```None```.

For example::

```
>>> m = re.match(r"(\d+)\.(\d+)", "24.1632")
>>> m.groups()
('24', '1632')
```

If we make the decimal place and everything after it optional, not all groups might participate in the match. These groups will default to ```None``` unless the **default** argument is given::

```
>>> m = re.match(r"(\d+)\.?(\d+)?", "24")
>>> m.groups()      # Second group defaults to None.
('24', None)
>>> m.groups('0')   # Now, the second group defaults to '0'.
('24', '0')
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1331)

Unknown directive type "method".

```
.. method:: Match.groupdict(default=None)
```

Return a dictionary containing all the **named** subgroups of the match, keyed by the subgroup name. The **default** argument is used for groups that did not participate in the match; it defaults to ```None```. For example::

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
>>> m.groupdict()
{'first_name': 'Malcolm', 'last_name': 'Reynolds'}
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1342)

Unknown directive type "method".

```
.. method:: Match.start([group])
           Match.end([group])
```

Return the indices of the start and end of the substring matched by **group**; **group** defaults to zero (meaning the whole matched substring). Return ```-1``` if **group** exists but did not contribute to the match. For a match object **m**, and a group **g** that did contribute to the match, the substring matched by group **g** (equivalent to ```m.group(g)```) is ::

```
m.string[m.start(g):m.end(g)]
```

Note that ```m.start(group)``` will equal ```m.end(group)``` if **group** matched a null string. For example, after ```m = re.search('b(c?)', 'cba')```, ```m.start(0)``` is 1, ```m.end(0)``` is 2, ```m.start(1)``` and ```m.end(1)``` are both 2, and ```m.start(2)``` raises an `:exc:IndexError` exception.

An example that will remove **remove_this** from email addresses::

```
>>> email = "tony@tiremove_thisger.net"
>>> m = re.search("remove_this", email)
>>> email[:m.start()] + email[m.end():]
'tony@tiger.net'
```


System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1366)

Unknown directive type "method".

```
.. method:: Match.span([group])
```

For a match `*m*`, return the 2-tuple `((m.start(group), m.end(group)))`. Note that if `*group*` did not contribute to the match, this is `((-1, -1))`. `*group*` defaults to zero, the entire match.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1373)

Unknown directive type "attribute".

```
.. attribute:: Match.pos
```

The value of `*pos*` which was passed to the `:meth:`~Pattern.search`` or `:meth:`~Pattern.match`` method of a `:ref:`regex object <re-objects>`. This is the index into the string at which the RE engine started looking for a match.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1380)

Unknown directive type "attribute".

```
.. attribute:: Match.endpos
```

The value of `*endpos*` which was passed to the `:meth:`~Pattern.search`` or `:meth:`~Pattern.match`` method of a `:ref:`regex object <re-objects>`. This is the index into the string beyond which the RE engine will not go.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1387)

Unknown directive type "attribute".

```
.. attribute:: Match.lastindex
```

The integer index of the last matched capturing group, or `None` if no group was matched at all. For example, the expressions `((a)b)`, `((a)(b))`, and `((ab))` will have `lastindex == 1` if applied to the string `'ab'`, while the expression `((a)(b))` will have `lastindex == 2`, if applied to the same string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1396)

Unknown directive type "attribute".

```
.. attribute:: Match.lastgroup
```

The name of the last matched capturing group, or `None` if the group didn't have a name, or if no group was matched at all.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1402)

Unknown directive type "attribute".

```
.. attribute:: Match.re
```

The `:ref:`regular expression object <re-objects>` whose `:meth:`~Pattern.match`` or

```
:meth:`~Pattern.search` method produced this match instance.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1408)

Unknown directive type "attribute".

```
.. attribute:: Match.string
```

The string passed to :meth:`~Pattern.match` or :meth:`~Pattern.search`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1413)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   Added support of :func:`~copy.copy` and :func:`~copy.deepcopy`. Match objects
   are considered atomic.
```

Regular Expression Examples

Checking for a Pair

In this example, we'll use the following helper function to display match objects a little more gracefully:

```
def displaymatch(match):
    if match is None:
        return None
    return '<Match: %r, groups=%r>' % (match.group(), match.groups())
```

Suppose you are writing a poker program where a player's hand is represented as a 5-character string with each character representing a card, "a" for ace, "k" for king, "q" for queen, "j" for jack, "t" for 10, and "2" through "9" representing the card with that value.

To see if a given string is a valid hand, one could do the following:

```
>>> valid = re.compile(r"^[a2-9tjqk]{5}$")
>>> displaymatch(valid.match("akt5q")) # Valid.
"<Match: 'akt5q', groups=()>"
>>> displaymatch(valid.match("akt5e")) # Invalid.
>>> displaymatch(valid.match("akt"))   # Invalid.
>>> displaymatch(valid.match("727ak")) # Valid.
"<Match: '727ak', groups=()>"
```

That last hand, "727ak", contained a pair, or two of the same valued cards. To match this with a regular expression, one could use backreferences as such:

```
>>> pair = re.compile(r"^(.*)\1$")
>>> displaymatch(pair.match("717ak")) # Pair of 7s.
"<Match: '717', groups=('7',)>"
>>> displaymatch(pair.match("718ak")) # No pairs.
>>> displaymatch(pair.match("354aa")) # Pair of aces.
"<Match: '354aa', groups=('a',)>"
```

To find out what card the pair consists of, one could use the :meth:`~Match.group` method of the match object in the following manner:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1460); [backlink](#)

Unknown interpreted text role "meth".

```
>>> pair = re.compile(r"^(.*)\1$")
>>> pair.match("717ak").group(1)
'7'
```

```
# Error because re.match() returns None, which doesn't have a group() method:
>>> pair.match("718ak").group(1)
Traceback (most recent call last):
```

```
File "<pyshell#23>", line 1, in <module>
    re.match(r".*(.)*\1", "718ak").group(1)
AttributeError: 'NoneType' object has no attribute 'group'

>>> pair.match("354aa").group(1)
'a'
```

Simulating scanf()

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1481)

Unknown directive type "index".

```
.. index:: single: scanf()
```

Python does not currently have an equivalent to `:c:func:scanf`. Regular expressions are generally more powerful, though also more verbose, than `:c:func:scanf` format strings. The table below offers some more-or-less equivalent mappings between `:c:func:scanf` format tokens and regular expressions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1483); [backlink](#)

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1483); [backlink](#)

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1483); [backlink](#)

Unknown interpreted text role "c:func".

`:c:func:scanf` Token

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1491); [backlink](#)

Unknown interpreted text role "c:func".

Regular Expression

<code>%c</code>	<code>.</code>
<code>%5c</code>	<code>.{5}</code>
<code>%d</code>	<code>[-+] ? \d +</code>
<code>%e, %E, %f, %g</code>	<code>[-+] ? (\d + (\. \d *) ? \. \d +) ([eE] [-+] ? \d +) ?</code>
<code>%i</code>	<code>[-+] ? (0 [xX] [\d A - F a - f] + 0 [0 - 7] * \d +)</code>
<code>%o</code>	<code>[-+] ? [0 - 7] +</code>
<code>%s</code>	<code>\\S +</code>
<code>%u</code>	<code>\\d +</code>
<code>%x, %X</code>	<code>[-+] ? (0 [xX]) ? [\d A - F a - f] +</code>

To extract the filename and numbers from a string like

```
/usr/sbin/sendmail - 0 errors, 4 warnings
```

you would use a `:c:func:scanf` format like

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1515); [backlink](#)

Unknown interpreted text role "c:func".

%s - %d errors, %d warnings

The equivalent regular expression would be

(\S+) - (\d+) errors, (\d+) warnings

search() vs. match()

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1529)

Unknown directive type "sectionauthor".

.. sectionauthor:: Fred L. Drake, Jr. <fdrake@acm.org>

Python offers two different primitive operations based on regular expressions: `func:re.match` checks for a match only at the beginning of the string, while `func:re.search` checks for a match anywhere in the string (this is what Perl does by default).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1531); [backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1531); [backlink](#)

Unknown interpreted text role "func".

For example:

```
>>> re.match("c", "abcdef") # No match
>>> re.search("c", "abcdef") # Match
<re.Match object; span=(2, 3), match='c'>
```

Regular expressions beginning with '^' can be used with `func:search` to restrict the match at the beginning of the string:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1542); [backlink](#)

Unknown interpreted text role "func".

```
>>> re.match("c", "abcdef") # No match
>>> re.search("^c", "abcdef") # No match
>>> re.search("^a", "abcdef") # Match
<re.Match object; span=(0, 1), match='a'>
```

Note however that in `const:MULTILINE` mode `func:match` only matches at the beginning of the string, whereas using `func:search` with a regular expression beginning with '^' will match at the beginning of each line.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1550); [backlink](#)

Unknown interpreted text role "const".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1550); [backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1550); [backlink](#)

Unknown interpreted text role "func".

```
>>> re.match('X', 'A\nB\nX', re.MULTILINE) # No match
>>> re.search('^X', 'A\nB\nX', re.MULTILINE) # Match
<re.Match object; span=(4, 5), match='X'>
```

Making a Phonebook

`:func:split` splits a string into a list delimited by the passed pattern. The method is invaluable for converting textual data into data structures that can be easily read and modified by Python as demonstrated in the following example that creates a phonebook.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1562); [backlink](#)

Unknown interpreted text role "func".

First, here is the input. Normally it may come from a file, here we are using triple-quoted string syntax

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1570)

Unknown directive type "doctest".

```
.. doctest::

>>> text = """Ross McFluff: 834.345.1254 155 Elm Street
... Ronald Heathmore: 892.345.3428 436 Finley Avenue
... Frank Burger: 925.541.7625 662 South Dogwood Way
...
... Heather Albrecht: 548.326.4584 919 Park Place"""
```

The entries are separated by one or more newlines. Now we convert the string into a list with each nonempty line having its own entry:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1583)

Unknown directive type "doctest".

```
.. doctest::
:options: +NORMALIZE_WHITESPACE

>>> entries = re.split("\n+", text)
>>> entries
['Ross McFluff: 834.345.1254 155 Elm Street',
 'Ronald Heathmore: 892.345.3428 436 Finley Avenue',
 'Frank Burger: 925.541.7625 662 South Dogwood Way',
 'Heather Albrecht: 548.326.4584 919 Park Place']
```

Finally, split each entry into a list with first name, last name, telephone number, and address. We use the `maxsplit` parameter of `:func:split` because the address has spaces, our splitting pattern, in it:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1593); [backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1597)

Unknown directive type "doctest".

```
.. doctest::
:options: +NORMALIZE_WHITESPACE

>>> [re.split("?: ", entry, 3) for entry in entries]
[['Ross', 'McFluff', '834.345.1254', '155 Elm Street'],
 ['Ronald', 'Heathmore', '892.345.3428', '436 Finley Avenue'],
 ['Frank', 'Burger', '925.541.7625', '662 South Dogwood Way'],
 ['Heather', 'Albrecht', '548.326.4584', '919 Park Place']]
```

The `: ?` pattern matches the colon after the last name, so that it does not occur in the result list. With a `maxsplit` of 4, we could separate the house number from the street name:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1610)

Unknown directive type "doctest".

```
.. doctest::
   :options: +NORMALIZE_WHITESPACE

>>> [re.split("?: ", entry, 4) for entry in entries]
[['Ross', 'McFluff', '834.345.1254', '155', 'Elm Street'],
 ['Ronald', 'Heathmore', '892.345.3428', '436', 'Finley Avenue'],
 ['Frank', 'Burger', '925.541.7625', '662', 'South Dogwood Way'],
 ['Heather', 'Albrecht', '548.326.4584', '919', 'Park Place']]
```

Text Munging

`:func:sub`` replaces every occurrence of a pattern with a string or the result of a function. This example demonstrates using `:func:sub`` with a function to "munge" text, or randomize the order of all the characters in each word of a sentence except for the first and last characters:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1623); [backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1623); [backlink](#)

Unknown interpreted text role "func".

```
>>> def repl(m):
...     inner_word = list(m.group(2))
...     random.shuffle(inner_word)
...     return m.group(1) + "".join(inner_word) + m.group(3)
>>> text = "Professor Abdolmalek, please report your absences promptly."
>>> re.sub(r"(\w)(\w+)(\w)", repl, text)
'Poefsrosr Aealmlodbk, pslaee reorpt your abnseces plmrptoy.'
>>> re.sub(r"(\w)(\w+)(\w)", repl, text)
'Pofsroser Aodlambelk, plasee reorpt yuor asnebes potlmrpy.'
```

Finding all Adverbs

`:func:findall`` matches *all* occurrences of a pattern, not just the first one as `:func:search`` does. For example, if a writer wanted to find all of the adverbs in some text, they might use `:func:findall`` in the following manner:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1642); [backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1642); [backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1642); [backlink](#)

Unknown interpreted text role "func".

```
>>> text = "He was carefully disguised but captured quickly by police."
>>> re.findall(r"\w+ly\b", text)
['carefully', 'quickly']
```

Finding all Adverbs and their Positions

If one wants more information about all matches of a pattern than the matched text, `:func:finder`` is useful as it provides `ref:match objects <match-objects>` instead of strings. Continuing with the previous example, if a writer wanted to find all of the adverbs *and their positions* in some text, they would use `:func:finder`` in the following manner:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1655); [backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1655); [backlink](#)

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\ (cpython-main) (Doc) (library) re.rst, line 1655); [backlink](#)

Unknown interpreted text role "func".

```
>>> text = "He was carefully disguised but captured quickly by police."
>>> for m in re.finditer(r"\w+ly\b", text):
...     print('%02d-%02d: %s' % (m.start(), m.end(), m.group(0)))
07-16: carefully
40-47: quickly
```

Raw String Notation

Raw string notation (`r"text"`) keeps regular expressions sane. Without it, every backslash (`'\'`) in a regular expression would have to be prefixed with another one to escape it. For example, the two following lines of code are functionally identical:

```
>>> re.match(r"\W(.)\1\W", " ff ")
<re.Match object; span=(0, 4), match=' ff '>
>>> re.match("\\W(.)\\1\\W", " ff ")
<re.Match object; span=(0, 4), match=' ff '>
```

When one wants to match a literal backslash, it must be escaped in the regular expression. With raw string notation, this means `r"\"`. Without raw string notation, one must use `"\\\"`, making the following lines of code functionally identical:

```
>>> re.match(r"\\", r"\\")
<re.Match object; span=(0, 1), match='\\'>
>>> re.match("\\\\", r"\\")
<re.Match object; span=(0, 1), match='\\'>
```

Writing a Tokenizer

A [tokenizer](#) or [scanner](#) analyzes a string to categorize groups of characters. This is a useful first step in writing a compiler or interpreter.

The text categories are specified with regular expressions. The technique is to combine those into a single master regular expression and to loop over successive matches:

```
from typing import NamedTuple
import re

class Token(NamedTuple):
    type: str
    value: str
    line: int
    column: int

def tokenize(code):
    keywords = {'IF', 'THEN', 'ENDIF', 'FOR', 'NEXT', 'GOSUB', 'RETURN'}
    token_specification = [
        ('NUMBER', r'\d+(\.\d*)?'), # Integer or decimal number
        ('ASSIGN', r':='),          # Assignment operator
        ('END', r';'),              # Statement terminator
        ('ID', r'[A-Za-z]+'),       # Identifiers
        ('OP', r'[+ \-*/]'),        # Arithmetic operators
        ('NEWLINE', r'\n'),         # Line endings
        ('SKIP', r'[ \t]+'),        # Skip over spaces and tabs
        ('MISMATCH', r'.'),         # Any other character
    ]
    tok_regex = '|'.join('%s' % pair for pair in token_specification)
    line_num = 1
    line_start = 0
    for mo in re.finditer(tok_regex, code):
        kind = mo.lastgroup
        value = mo.group()
        column = mo.start() - line_start
        if kind == 'NUMBER':
```



```

        value = float(value) if '.' in value else int(value)
    elif kind == 'ID' and value in keywords:
        kind = value
    elif kind == 'NEWLINE':
        line_start = mo.end()
        line_num += 1
        continue
    elif kind == 'SKIP':
        continue
    elif kind == 'MISMATCH':
        raise RuntimeError(f'{value!r} unexpected on line {line_num}')
    yield Token(kind, value, line_num, column)

statements = '''
    IF quantity THEN
        total := total + price * quantity;
        tax := price * 0.05;
    ENDIF;
'''

for token in tokenize(statements):
    print(token)

```

The tokenizer produces the following output:

```

Token(type='IF', value='IF', line=2, column=4)
Token(type='ID', value='quantity', line=2, column=7)
Token(type='THEN', value='THEN', line=2, column=16)
Token(type='ID', value='total', line=3, column=8)
Token(type='ASSIGN', value=':=', line=3, column=14)
Token(type='ID', value='total', line=3, column=17)
Token(type='OP', value='+', line=3, column=23)
Token(type='ID', value='price', line=3, column=25)
Token(type='OP', value='*', line=3, column=31)
Token(type='ID', value='quantity', line=3, column=33)
Token(type='END', value=';', line=3, column=41)
Token(type='ID', value='tax', line=4, column=8)
Token(type='ASSIGN', value=':=', line=4, column=12)
Token(type='ID', value='price', line=4, column=15)
Token(type='OP', value='*', line=4, column=21)
Token(type='NUMBER', value=0.05, line=4, column=23)
Token(type='END', value=';', line=4, column=27)
Token(type='ENDIF', value='ENDIF', line=5, column=4)
Token(type='END', value=';', line=5, column=9)

```

[Frie09] Friedl, Jeffrey. *Mastering Regular Expressions*. 3rd ed., O'Reilly Media, 2009. The third edition of the book no longer covers Python at all, but the first edition covered writing good regular expression patterns in great detail.