re — Regular expression operations

(Pattern Matching)

re.**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 **MatchObject** instance. Return `None` if no position in the string matches the pattern;

re.**match**(*pattern*, *string*, *flags=0*)

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

re.**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.

Example Program

```
#!/usr/bin/python

import re

str = 'hello'
found  = re.search(r'll', str)
if found:
    print 'found', found.group()
else:
    print 'did not find'
```

Regular Expression Syntax

Quantifiers and special characters are:

`.`

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

`^`

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

`$`

Matches the end of the string or just before the newline at the end of the string, and in **MULTILINE** mode also matches before a newline.

`*`

Matches 0 or more repetitions of the preceding.

`+`

Matches 1 or more repetitions of the preceding.

`?`

Matches 0 or 1 repetitions of the preceding.

`*?`, `+?`, `??`

The `'*'`, `'+'`, and `'?'` qualifiers are all *greedy*; they match as much text as possible. Adding `'?'` after the qualifier makes it perform the match in *non-greedy* or *minimal* fashion; as *few* characters as possible will be matched.

`{m}`

Specifies that exactly *m* copies of the previous

`{m,n}`

Match from *m* to *n* repetitions of the preceding

`{m,n}?`

Causes the resulting RE to match from *m* to *n* repetitions of the preceding, attempting to match as *few* repetitions as possible.

\

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

[]

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

- Characters can be listed individually, e.g. [abc] will match 'a', 'b', or 'c'.
- 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.
- Character classes such as \w or \S (defined below) are also accepted inside a set.
- Characters that are not within a range can be matched by *complementing* the set. If the first character of the set is '^', all the characters that are *not* in the set will be matched.

'|'

A|B creates a regular expression that will match either A or B. An arbitrary number of REs can be separated by the '|' in this way.

(?=...)

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

(?!...)

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

(?<=...)

Matches if the current position in the string is preceded by a match for ... that ends at the current position. This is called a *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. Group references are not supported even if they match strings of some fixed length. 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 **search()**function rather than the **match()** function:

```
>>>
```

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

'def'

This example looks for a word following a hyphen:

```
>>>
```

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

(?<!...)

Matches if the current position in the string is not preceded by a match for .... This is called a *negative lookbehind assertion*.

\A

Matches only at the start of the string.

\b

Matches the empty string, but only at the beginning or end of a word. (word boundary).

`\B`

Matches the empty string, but only when it is *not* at the beginning or end of a word. (not a word boundary).

`\d`

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

`\D`

Matches any non-digit character; this is equivalent to the set `[^0-9]`.

`\s`

Matches any whitespace character, this is equivalent to the set `[ \t\n\r\f\v]`.

`\S`

Matches any non-whitespace character; this is equivalent to the set `[^ \t\n\r\f\v]` .

`\w`

When the **LOCALE** and **UNICODE** flags are not specified, matches any alphanumeric character and the underscore; this is equivalent to the set`[a-zA-Z0-9_]`. With **LOCALE**, it will match the set `[0-9_]` plus whatever characters are defined as

alphanumeric for the current locale. If **UNICODE** is set, this will match the characters [0-9_] plus whatever is classified as alphanumeric in the Unicode character properties database.

\W

When the **LOCALE** and **UNICODE** flags are not specified, matches any non-alphanumeric character; this is equivalent to the set [^a-zA-Z0-9_]. With **LOCALE**, it will match any character not in the set [0-9_], and not defined as alphanumeric for the current locale. If **UNICODE** is set, this will match anything other than [0-9_] plus characters classified as not alphanumeric in the Unicode character properties database.

\Z

Matches only at the end of the string.