A *regular expression* is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. The module ***re*** provides full support for Perl-like regular expressions in Python.

## The *match* Function

This function attempts to match RE *pattern* to *string* with optional *flags*:

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

## The *search* Function

This function search for first occurrence of RE *pattern* within *string* with optional *flags*:

re.search(pattern, string, flags=0)

Here is the description of the parameters:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| pattern | This is the regular expression to be matched. |
| string | This is the string which would be searched to match the pattern at the beginning of string. |
| flags | You can specifiy different flags using bitwise OR (|). |

## The *search* Function

This function search for first occurrence of RE *pattern* within *string* with optional *flags*:

re.search(pattern, string, flags=0)

*re.match()* and *re.search()* returns a **match** object on success, **None** on failure. Use *group(num)* or *groups()* of **match** object to get matched expression.

|  |  |
| --- | --- |
| **Match Object Methods** | **Description** |
| group(num=0) | This methods returns entire match (or specific subgroup num) |
| groups() | This method return all matching subgroups in a tuple (empty if there weren't any) |

## Matching vs Searching:

Python offers two different primitive operations based on regular expressions: **match** checks for a match only at the beginning of the string, while **search** checks for a match anywhere in the string.

### EXAMPLE:

*#!/usr/bin/python*

*import re*

*line = "Cats are smarter than dogs";*

*matchObj = re.match( r'dogs', line, re.M|re.I)*

*if matchObj:*

*print "match --> matchObj.group() : ", matchObj.group()*

*else:*

*print "No match!!"*

*matchObj = re.search( r'dogs', line, re.M|re.I)*

*if matchObj:*

*print "search --> matchObj.group() : ", matchObj.group()*

*else:*

*print "No match!!"*

When the above code is executed, it produces following result:

*No match!!*

*search --> matchObj.group() : dogs*

## Search and Replace

Some of the most important **re** methods that use regular expressions is **sub**.

*re.sub(pattern, repl, string, max=0)*

This method replace all occurrences of the RE *pattern* in *string* with *repl*, substituting all occurrences unless *max* provided. This method would return modified string.

### EXAMPLE:

Following is the example:

*#!/usr/bin/python*

*phone = "2004-959-559 #This is Phone Number"*

*# Delete Python-style comments*

*num = re.sub(r'#.\*$', "", phone)*

*print "Phone Num : ", num*

*# Remove anything other than digits*

*num = re.sub(r'\D', "", phone)*

*print "Phone Num : ", num*

When the above code is executed, it produces following result:

Phone Num : 2004-959-559

Phone Num : 2004959559

## Regular-expression Modifiers - Option Flags

Regular expression may include an optional modifier to control various aspects of matching. The modifiers are specified as an optional flag:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| re.I | Performs case-insensitive matching. |
| re.L | Interprets words according to the current locale.This interpretation affects the alphabetic group (\w and \W), as well as word boundary behavior (\b and \B). |
| re.M | Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string). |
| re.S | Makes a period (dot) match any character, including a newline. |
| re.U | Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B. |
| re.X | Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [] or when escaped by a backslash), and treats unescaped # as a comment marker. |

## Regular-expression patterns

Except for control characters, **(+ ? . \* ^ $ ( ) [ ] { } | \)**, all characters match themselves. You can escape a control character by preceding it with a backslash. Following table lists the regular expression syntax that is available in Python.

|  |  |
| --- | --- |
| **Pattern** | **Description** |
| ^ | Matches beginning of line. |
| $ | Matches end of line. |
| . | Matches any single character except newline. |
| [...] | Matches any single character in brackets. |
| [^...] | Matches any single character not in brackets |
| re\* | Matches 0 or more occurrences of preceding expression. |
| re+ | Matches 1 or more occurrence of preceding expression. |
| re? | Matches 0 or 1 occurrence of preceding expression. |
| re{ n} | Matches exactly n number of occurrences of preceding expression. |
| re{ n,} | Matches n or more occurrences of preceding expression. |
| re{ n, m} | Matches at least n and at most m occurrences of preceding expression. |
| a| b | Matches either a or b. |
| (re) | Groups regular expressions and remembers matched text. |
| (?imx) | Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected. |
| (?-imx) | Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected. |
| (?: re) | Groups regular expressions without remembering matched text. |
| (?imx: re) | Temporarily toggles on i, m, or x options within parentheses. |
| (?-imx: re) | Temporarily toggles off i, m, or x options within parentheses. |
| (?#...) | Comment. |
| (?= re) | Specifies position using a pattern. Doesn't have a range. |
| (?! re) | Specifies position using pattern negation. Doesn't have a range. |
| (?> re) | Matches independent pattern without backtracking. |
| \w | Matches word characters. |
| \W | Matches nonword characters. |
| \s | Matches whitespace. Equivalent to [\t\n\r\f]. |
| \S | Matches nonwhitespace. |
| \d | Matches digits. Equivalent to [0-9]. |
| \D | Matches nondigits. |

# Regular-expression Examples

## Literal characters:

|  |  |
| --- | --- |
| **Example** | **Description** |
| python | Match "python". |

## Character classes:

|  |  |
| --- | --- |
| **Example** | **Description** |
| [Pp]ython | Match "Python" or "python" |
| rub[ye] | Match "ruby" or "rube" |
| [aeiou] | Match any one lowercase vowel |
| [0-9] | Match any digit; same as [0123456789] |
| [a-z] | Match any lowercase ASCII letter |
| [A-Z] | Match any uppercase ASCII letter |
| Y | Match any of the above |
| [^aeiou] | Match anything other than a lowercase vowel |
| [^0-9] | Match anything other than a digit |

## Special Character Classes:

|  |  |
| --- | --- |
| **Example** | **Description** |
| . | Match any character except newline |
| \d | Match a digit: [0-9] |
| \D | Match a nondigit: [^0-9] |
| \s | Match a whitespace character: [ \t\r\n\f] |
| \S | Match nonwhitespace: [^ \t\r\n\f] |
| \w | Match a single word character: [A-Za-z0-9\_] |
| \W | Match a nonword character: [^A-Za-z0-9\_] |

## Repetition Cases:

|  |  |
| --- | --- |
| **Example** | **Description** |
| ruby? | Match "rub" or "ruby": the “y” is optional |
| ruby\* | Match "rub" plus 0 or more “y”s |
| ruby+ | Match "rub" plus 1 or more “y”s |
| \d{3} | Match exactly 3 digits |
| \d{3,} | Match 3 or more digits |
| \d{3,5} | Match 3, 4, or 5 digits |

## Alternatives:

|  |  |
| --- | --- |
| **Example** | **Description** |
| python|perl | Match "python" or "perl" |
| rub(y|le) | Match "ruby" or "ruble" |
| Python(!+|\?) | "Python" followed by one or more ! or one ? |

## Anchors:

This need to specify match position

|  |  |
| --- | --- |
| **Example** | **Description** |
| ^Python | Match "Python" at the start of a string or internal line |
| Python$ | Match "Python" at the end of a string or line |