Regular Expressions by Example

Python Specific Examples

This section is lifted from Wikipedia's Regular expression examples page. All I've done is translate the code from Perl to Python 2. For this version, assume that the "`import re`" command has already been issued.

Metacharacter(s)	Description	Example Note that all the if statements return a TRUE value

.	Normally matches any character except a newline. Within square brackets the dot is literal.	`string1 = "Hello, world."`
		`if re.search(r".....", string1):`
		`  print string1 + " has length >= 5"`

( )	Groups a series of pattern elements to a single element. When you match a pattern within parentheses, you can use any of $1, $2, ... later to refer to the previously matched pattern.	`string1 = "Hello, world."`
		`m_obj = re.search(r"(H..).(o..)", string1)`
		`if m_obj:`
		`  print "We matched '" + m_obj.group(1) +\`
		`  "' and '" + m_obj.group(2) + "'"`
		Output:
		`We matched 'Hel' and 'o, ';`

+	Matches the preceding pattern element one or more times.	`string1 = "Hello, world."`
		`if re.search(r"l+", string1):`
		`  print 'There are one or more consecutive letter "l"' +\`
		`  "'s in " + string1`
		Output:
		`There are one or more consecutive letter "l"'s in Hello World`

?	Matches the preceding pattern	`string1 = "Hello, world."`
		`if re.search(r"H.?e", string1):`
		`  print "There is an 'H' and a 'e' separated by " +\`

	element zero or one times.	`"0-1 characters (Ex: He Hoe)\n"`

?	Modifies the *, +, or {M,N}'d regexp that comes before to match as few times as possible.	`string1 = "Hello, world."`
		`if re.search(r"l.+?o", string1):`
		`  print "The non-greedy match with 'l' followed by\n" +\`
		`  "one or more characters is 'llo' rather than\n" +\`
		`  "'llo wo'."`

*****	Matches the preceding pattern element zero or more times.	`string1 = "Hello, world."`
		`if re.search(r"el*o", string1):`
		`  print "There is an 'e' followed by zero to many\n" +\`
		`  "'l' followed by 'o' (eo, elo, ello, elllo)"`

{M,N}	Denotes the minimum M and the maximum N match count.	`string1 = "Hello, world."`
		`if re.search(r"l{1,2}", string1):`
		`  print "There exists a substring with at least 1\n" +\`
		`  "and at most 2 l's in " + string1`

[...]	Denotes a set of possible character matches.	`string1 = "Hello, world."`
		`if re.search(r"[aeiou]+", string1):`
		`  print string1 + " contains one or more vowels."`

|	Separates alternate possibilities.	`string1 = "Hello, world."`
		`if re.search(r"(Hello|Hi|Pogo)", string1):`
		`  print "At least one of Hello, Hi, or Pogo is " +\`
		`  "contained in " + string1`

\b	Matches a word boundary.	`string1 = "Hello World"`
		`if re.search(r"llo\b", string1):`
		`  print "There is a word that ends with 'llo'"`
		`else:`
		`  print "There are no words that end with 'llo'"`

\w	Matches an alphanumeric character, including "_".	`string1 = "Hello World"`
		`m_obj = re.search(r"(\w\w)", string1)`
		`if m_obj:`
		`  print "The first two adjacent alphanumeric characters"`
		`  print "(A-Z, a-z, 0-9, _) in", string1, "were",`
		`  print m_obj.group(1)`
		Output:
		`The first two adjacent alphanumeric characters`
		`(A-Z, a-z, 0-9, _) in Hello World were He`

\W	Matches a non-alphanumeric character,	`string1 = "Hello World"`
		`if re.search(r"\W", string1):`
		`  print "The space between Hello and " +\`

	excluding "_".	"World is not alphanumeric"

\s	Matches a whitespace character (space, tab, newline, form feed)	`string1 = "Hello World\n"`
		`if re.search(r"\s.*\s", string1):`
		`  print "There are TWO whitespace characters, which may"`
		`  print "be separated by other characters, in", string1`

\S	Matches anything BUT a whitespace.	`string1 = "Hello World\n"`
		`m_obj = re.search(r"(\S*)\s*(\S*)", string1)`
		`if m_obj:`
		`  print "The first two groups of NON-whitespace characters"`
		`  print "are '%s' and '%s'." % m_obj.groups()`
		Output:
		`The first two groups of NON-whitespace characters`
		`are 'Hello' and 'World'.`

\d	Matches a digit, same as [0-9].	`string1 = "99 bottles of beer on the wall."`
		`m_obj = re.search(r"(\d+)", string1)`
		`if m_obj:`
		`  print m_obj.group(1), "is the first number in '" +\`
		`  string1 + "'"`
		Output:
		`99 is the first number in '99 bottles of beer on the wall.'`

\D	Matches a non-digit.	`string1 = "Hello World"`
		`if re.search(r"\D", string1):`
		`  print "There is at least one character in", string1,`
		`  print "that is not a digit."`

^	Matches the beginning of a line or string.	`string1 = "Hello World"`
		`if re.search(r"^He", string1):`
		`  print string1, "starts with the characters 'He'"`

$	Matches the end of a line or string.	`string1 = "Hello World"`
		`if re.search(r"rld$", string1):`
		`  print string1, "is a line or string " +\`
		`  "that ends with 'rld'"`

\A	Matches the beginning of a string (but not an internal line).	`string1 = "Hello\nWorld\n"`
		`if re.search(r"\AH", string1):`
		`  print string1, "is a string",`
		`  print "that starts with 'H'"`
		Output:
		`Hello`
		`World`

		`is a string that starts with 'H'`

\Z	Matches the end of a string (but not an internal line).	`string1 = "Hello\nWorld\n"`
		`if re.search(r"d\n\Z", string1):`
		`  print string1, "is a string",`
		`  print "that ends with 'd\\n'"`

[^...]	Matches every character except the ones inside brackets.	`string1 = "Hello World\n"`
		`if re.search(r"[^abc]", string1):`
		`  print string1 + " contains a character other than " +\`
		`  "a, b, and c"`

Other Examples

In this section I will present different regular expressions followed by strings with the area that matched highlighted in yellow, followed by any commentary. In order to make it clear when there are any spaces at the end of a multiline string, the background of the nonmatched portions of the strings are set to white.

I'll probably add examples here in the future as they occure to me or I see them elsewhere.

Regex: **ATG**

String:

```
TGCGTAGGCCAGACGGCAGTCCCGCGGCGGGATCCTGGTAAGGTGAAATA
TGGGACACGCAAATTGGCGGCAGGGGCAACGCTACTACGGCAAACACACT
TAGACAGACGTGACATACTTCATCATGTCGTTCGGGGCCATAATTCCCCG
ACGCTGAGTAGATCAATGTATCATAGGTTTACGTCATTTCAATCTAGTGC
CGGGTTAGCTGACTACGACCCCCTGCCCCTCTAACGTGTGGGTAATCTTG
```

Regex: **A\s?T\s?G**

String:

```
TGCGTAGGCCAGACGGCAGTCCCGCGGCGGGATCCTGGTAAGGTGAAATA
TGGGACACGCAAATTGGCGGCAGGGGCAACGCTACTACGGCAAACACACT
TAGACAGACGTGACATACTTCATCATGTCGTTCGGGGCCATAATTCCCCG
ACGCTGAGTAGATCAATGTATCATAGGTTTACGTCATTTCAATCTAGTGC
CGGGTTAGCTGACTACGACCCCCTGCCCCTCTAACGTGTGGGTAATCTTG
```

Comments: Notice how the second regex matches an ATG much earlier in the sequence. The first ATG is interupted by a newline, but the second regular expression is tolerant of whitespace in between the letters it is looking for. This is an ok technique for small search strings, but it can get cumbersome for larger strings. To allow for multiple whitespace characters, use `\s*` instead of `\s?`

Regex: **CG\S|AG(A|G)**

String:

```
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA ATC ATG ATT
CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC CGG CGT CTA CTC CTG CTT
GAA GAC GAG GAT GCA GCC GCG GCT GGA GGC GGG GGT GTA GTC GTG GTT
TAA TAC TAG TAT TCA TCC TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
```

Comments: Those are the six DNA codons that code for Arginine. this expression illustrates the global nature of the '|' metacharacter. The second '|' is not necessary, as it is separating single characters. It would work exactly the same way if we used **`CG\S|AG[AG]`** instead.