Regular Expressions

Boo has built-in support for regular expression literals.

You surround the regular expression with / /,
or @/ / for more complex expressions that contain whitespace.
Boo even has an =~ operator like Perl.

Here are some samples of using regular expressions in boo:

```
//pattern matching using perl's match operator (=~)
samplestring = "Here is foo"
if samplestring =~ /foo/:
       print "it's a match"

//sample regex object
re = /foo(bar)/
print re.GetType() //-> System.Text.RegularExpressions.Regex

//split a line on spaces:
words = @/ /.Split(samplestring)  //you can also use \s for any type of whitespace
print join(words, ",")

//similar example with unpacking
s = "First Last"
first, last = @/ /.Split(s)

//another way to do matching without =~
m = /abc/.Match("123abc456")
if m.Success:
      print "Found match at position:", m.Index
   print "Matched text:", m.ToString()

//more complex example with named groups
s = """
Joe Jackson
131 W. 5th Street
New York, NY  10023
"""

r =
/(?<=\n)\s*(?<city>[^\n]+)\s*,\s*(?<state>\w+)\s+(?<zip>\d{5}(-\d{4})?).*$/.Match(s)

print r.Groups["city"]
print r.Groups["state"]
print r.Groups["zip"]

//just for reference, the match operator (=~) works with regular strings, too:
if samplestring =~ "foo":
        print "it matches"

//the "not match" operator (!~) has not been implemented yet, but you
//can simply prefix the expression with "not":
if not samplestring =~ /badfoo/:
       print "no match"
```

Specifying regex options

One limitation of using built-in support for regular expressions is that you can't use non-standard regex options like regex compiled.

Actually there is a way to use the ignore case option. Add a (?i) to the beginnning of your regex pattern like so:

```
pattern = /(?i)google/
print "google" =~ pattern
print "Google" =~ pattern
```

But in other cases you may want to just use the .NET Regex class explicitly:

```
import System.Text.RegularExpressions

samplestring = "Here is foo"
re = Regex("FOO", RegexOptions.IgnoreCase | RegexOptions.Compiled)

//one way:
if samplestring =~ re:
   print "we matched"

//another using Match
if re.IsMatch(samplestring):
 print "we matched"
```

Regex Replace method

This would be an equivalent to using perl's switch/replace statement: s/foo/bar/g.

```
import System.Text.RegularExpressions

text = "four score and seven years ago"
print text

//replace each word with "X"
t2 = /\w+/.Replace(text, "X")
print t2 //-> X X X X X X

//replace only the first occurence of a word with X,
 // starting after the 15th character:
t3 = /\w+/.Replace(text, "X", 1, 15)
print t3 //-> four score and X years ago

//use a closure (or a regular method) to capitalize each word:
t4 = /\w+/.Replace(text) do (m as Match):
    s = m.ToString()
    if System.Char.IsLower(s[0]): //built-in char type will be added soon
         return System.Char.ToUpper(s[0]) + s[1:]
    return s
print t4 //-> Four Score And Seven Years Ago


//Back References are supported too! using the dollar sign
phonenumber = "5551234567"
phonenumber = /(\d{3})(\d{3})(\d{4})/.Replace(phonenumber, "($1) $2-$3")
print phonenumber //-> (555) 123-4567
```

regex primitive type

Also note, Boo has a built-in primitive type called "regex" (lowercase) that means the same thing as the .NET Regex class. So you can do for example:

```
import System.Text.RegularExpressions

re = /foo(bar)/

if re isa regex:
    print "re is a regular expression"

//or declare the type explicitly:
re2 as regex = /foo(bar)/
print re2 isa regex

//using the regex primitive constructor
//(You'll need an import statement for RegexOptions)
re3 = regex("FOO", RegexOptions.IgnoreCase | RegexOptions.Compiled)
print re3 isa regex
```

See also:

- Using Regular Expressions in .NET
- .NET Framework Regular Expressions
- The Regular Expression Library has hundreds of user-contributed regex samples.
- txt2regex - command line to assist with constructing regular expressions