Note in the last example, the end of the string is considered a word boundary.
You might wonder why '.' matches everything but "\n" - why not every character? The reason is that often one is matching against lines and would like to ignore the newline characters. For instance, while the string "\n" represents one line, we would like to think of as empty. Then

```
""   =~ /^$/;    # matches
"\n" =~ /^$/;    # matches, "\n" is ignored

""   =~ /./;      # doesn't match; it needs a char
""   =~ /^.$/;    # doesn't match; it needs a char
"\n" =~ /^.$/;    # doesn't match; it needs a char other than "\n"
"a"  =~ /^.$/;    # matches
"a\n"  =~ /^.$/;  # matches, ignores the "\n"
```

This behavior is convenient, because we usually want to ignore newlines when we count and match characters in a line. Sometimes, however, we want to keep track of newlines. We might even want ^ and $ to anchor at the beginning and end of lines within the string, rather than just the beginning and end of the string. Perl allows us to choose between ignoring and paying attention to newlines by using the //s and //m modifiers. //s and //m stand for single line and multi-line and they determine whether a string is to be treated as one continuous string, or as a set of lines. The two modifiers affect two aspects of how the regexp is interpreted: 1) how the '.' character class is defined, and 2) where the anchors ^ and $ are able to match. Here are the four possible combinations:

- no modifiers (//): Default behavior. '.' matches any character except "\n". ^ matches only at the beginning of the string and $ matches only at the end or before a newline at the end.
- s modifier (//s): Treat string as a single long line. '.' matches any character, even "\n". ^ matches only at the beginning of the string and $ matches only at the end or before a newline at the end.
- m modifier (//m): Treat string as a set of multiple lines. '.' matches any character except "\n". ^ and $ are able to match at the start or end of *any* line within the string.
- both s and m modifiers (//sm): Treat string as a single long line, but detect multiple lines. '.' matches any character, even "\n". ^ and $, however, are able to match at the start or end of *any* line within the string.

Here are examples of //s and //m in action:

```
$x = "There once was a girl\nWho programmed in Perl\n";

$x =~ /^Who/;   # doesn't match, "Who" not at start of string
$x =~ /^Who/s;  # doesn't match, "Who" not at start of string
$x =~ /^Who/m;  # matches, "Who" at start of second line
$x =~ /^Who/sm; # matches, "Who" at start of second line

$x =~ /girl.Who/;   # doesn't match, "." doesn't match "\n"
$x =~ /girl.Who/s;  # matches, "." matches "\n"
$x =~ /girl.Who/m;  # doesn't match, "." doesn't match "\n"
$x =~ /girl.Who/sm; # matches, "." matches "\n"
```

Most of the time, the default behavior is what is want, but //s and //m are occasionally very useful. If //m is being used, the start of the string can still be matched with \A and the end of string can still be matched with the anchors \Z (matches both the end and the newline before, like $), and \z (matches only the end):

```
$x =~ /^Who/m;   # matches, "Who" at start of second line
$x =~ /\AWho/m;  # doesn't match, "Who" is not at start of string

$x =~ /girl$/m;  # matches, "girl" at end of first line
$x =~ /girl\Z/m; # doesn't match, "girl" is not at end of string

$x =~ /Perl\Z/m; # matches, "Perl" is at newline before end
$x =~ /Perl\z/m; # doesn't match, "Perl" is not at end of string
```

We now know how to create choices among classes of characters in a regexp. What about choices among words or character strings? Such choices are described in the next section.