The following examples cover some of the most common forms of regular expressions:

Regular Expression	**Matches**
`joy`	Any string containing the character `j`, followed by an `o` and a `y`. It thus matches `joy`, and `enjoy`, among many others. `Joyful`, however, does not match as it contains an uppercase `J`.
`[Jj]oy`	Any string that starts with an upper-case `J` *or* a lower-case `j` and is followed by `o` and `y`. Matches for example the strings `Joy`, `joy`, `enjoy`, and `enJoy`.
`[0-9]`	Any single digit from 0 to 9.
`[a-zA-Z]`	Any single letter in the range a-z, whether upper- or lower-case.
`^`	Start of a string.
`^Host`	`Host` when it is found at the start of a string.
`$`	End of a string.
`^Host$`	A string containing only the word `Host`.
`. (dot)`	Any character.
`p.t`	`pat`, `pet`, and `pzt`, among others.

Regular expressions can contain *metacharacters*. We have already seen an example of these in the table above: `^`, `$`, and "dot" don't match any one character but have other meaning within regular expressions (start of string, end of string and match any character in this case). The following table lists some additional metacharacters that are frequently used in regexes:

Metacharacter	**Meaning**
`*`	Match the preceding character or sequence 0 or more times.
`?`	Match the preceding character or sequence 0 or 1 times.
`+`	Match the preceding character or sequence 1 or more times.

So for example if we wanted to match `favorite` or `favourite`, we could use the regex `favou?rite`. Similarly, if we wanted to match either `previous` or `previously` we could use the regex `previous(ly)?`. The parentheses— `()` —group the `ly` characters and then apply the `?` operator to the group to match it 0 or 1 times (therefore making it optional).

So what if we really do want to match a dot literally, and not have it interpreted as any character? In that case we need to *escape* the dot with a backslash. Referring back to our previous example, if we really did want to match the string `p.t` literally, we would use the regex `p\.t` to ensure that the dot is interpreted like a literal character and not a metacharacter by the regex engine.