Unfortunately, it also matches “I am the egg man. I am the egg man.”. (What kind of sense does that make?) More importantly, it also matches “I am the walrus. I am the egg man. I am the walrus.”, even though the number of repetitions is explicitly capped at two. That’s because the pattern need not match the entire search text. Here, the regex matches two sentences and terminates, declaring success. It simply doesn’t care that another repetition is available.

It is a common error to confuse the regular expression metacharacter * (the zero-or-more quantifier) with the shell’s * globbing character. The regex version of the star needs something to modify; otherwise, it won’t do what you expect. Use .* if any sequence of characters (including no characters at all) is an acceptable match.

Example regular expressions

In the United States, postal (“zip”) codes have either five digits or five digits followed by a dash and four more digits. To match a regular zip code, you must match a five-digit number. The following regular expression fits the bill:

```
^\d{5}$
```

The ^ and $ match the beginning and end of the search text but do not actually correspond to characters in the text; they are “zero-width assertions.” These characters ensure that only texts consisting of exactly five digits match the regular expression—the regex will not match five digits within a larger string. The `\d` escape matches a digit, and the quantifier `{5}` says that there must be exactly five digit matches.

To accommodate either a five-digit zip code or an extended zip+4, add an optional dash and four additional digits:

```
^\d{5}(-\d{4})?$
```

The parentheses group the dash and extra digits together so that they are considered one optional unit. For example, the regex won’t match a five-digit zip code followed by a dash. If the dash is present, the four-digit extension must be present as well or there is no match.

A classic demonstration of regex matching is the following expression,

```
M[ou]'?am+[ae]r ([AEae]l[- ])?[GKQ]h?[aeu]+([dtz][dhz]?){1,2}af[iy]
```

which matches most of the variant spellings of the name of Libyan head of state Moammar Gadhafi, including

• Muammar al-Kaddafi	(BBC)
• Moammar Gadhafi	(Associated Press)
• Muammar al-Qadhafi	(Al-Jazeera)
• Mu’ammar Al-Qadhafi	(U.S. Department of State)

Do you see how each of these would match the pattern?

This regular expression also illustrates how quickly the limits of legibility can be reached. Many regex systems (including Perl's) support an `x` option that ignores literal whitespace in the pattern and enables comments, allowing the pattern to be spaced out and split over multiple lines. You can then use whitespace to separate logical groups and clarify relationships, just as you would in a procedural language. For example:

```
M [ou] '? a m+ [ae] r   # First name: Mu'ammar, Moamar, etc.
\s                      # Whitespace; can't use a literal space here
(                       # Group for optional last name prefix
  [AEae] l              #    Al, El, al, or el
  [-\s]                 #    Followed by dash or space
)?
[GKQ] h? [aeu]+         # Initial syllable of last name: Kha, Qua, etc.
(                       # Group for consonants at start of 2nd syllable
  [dtz] [dhz]?          #    dd, dh, etc.
){1,2}                  # Group might occur twice, as in Quadhdhafi
af [iy]
```

This helps a little bit, but it's still pretty easy to torture later readers of your code. So be kind: if you can, use hierarchical matching and multiple small matches instead of trying to cover every possible situation in one large regular expression.

Captures

When a match succeeds, every set of parentheses becomes a "capture group" that records the actual text that it matched. The exact manner in which these pieces are made available to you depends on the implementation and context. In Perl, you can access the results as a list or as a sequence of numbered variables.

Since parentheses can nest, how do you know which match is which? Easy—the matches arrive in the same order as the opening parentheses. There are as many captures as there are opening parentheses, regardless of the role (or lack of role) that each parenthesized group played in the actual matching. When a parenthesized group is not used (e.g., `Mu(')?ammar` when matched against "Muammar"), its corresponding capture is empty.

If a group is matched more than once, only the contents of the last match are returned. For example, with the pattern

```
(I am the (walrus|egg man)\. ?){1,2}
```

matching the text

```
I am the egg man. I am the walrus.
```

there are two results, one for each set of parentheses:

```
I am the walrus.
walrus
```