<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=~</td>
<td>Binding operator to match pattern against string&lt;br&gt;<strong>e.g.</strong> <code>if ($str =~ /$pattern/) { ... }</code></td>
</tr>
<tr>
<td>/</td>
<td>Commonly used pattern delimiters</td>
</tr>
<tr>
<td>/i</td>
<td>Case insensitive match</td>
</tr>
<tr>
<td></td>
<td>Alternation, e.g. `/C</td>
</tr>
<tr>
<td>[ ]</td>
<td>Character class, e.g. <code>[CG]</code> or negated class <code>[^ATN]</code></td>
</tr>
<tr>
<td>*</td>
<td>Match 0 or more occurrences of the previous element</td>
</tr>
<tr>
<td>+</td>
<td>Match 1 or more occurrences of the previous element</td>
</tr>
<tr>
<td>?</td>
<td>Match 0 or 1 occurrences of the previous element</td>
</tr>
<tr>
<td>{n,m}</td>
<td>Match $n$ to $m$ occurrences of the previous element</td>
</tr>
<tr>
<td>\d, \s</td>
<td>Match digit, whitespace character</td>
</tr>
</tbody>
</table>
So far we have learned about the binding operator =~ for pattern matching, the /i modifier for case insensitive matches, character classes, and quantifiers. What is matched and what is output by the following code?

```perl
$seq = "AAACAGCAGCAGCAGCAGTTTNNT";
$cag = "cag";
if ($seq =~ /$cag/) { print "Found $cag \n"; }
if ($seq =~ /$cag/i) { print "Found $cag mixed case \n"; }
if ($seq =~ /T+/) { print "Found at least one T \n"; }
if ($seq =~ /[GC]/) { print "Found G or C \n"; }
if ($seq =~ /[^ACGT]/) { print "Found unknown base \n"; }
if ($seq =~ /T{2,3}/) { print "Found 2-3 consecutive T's \n"; }
if ($seq =~ /A{10,}/) { print "Found at least 10 A's \n"; }
```

What are two other ways to write the following pattern?

```regex
/0|1|2|3|4|5|6|7|8|9/
```
in grep

• use the -P for perl syntaxed regex
  • `grep -P "<search>" <file>`
Anchored Matches

• If you want to require a match to be at the beginning or end of a line, it is possible to anchor the match using the ^ and/or $ characters. Examples:

  • $seq =~ /^CAG/;  # match CAG at beginning of $seq
  • $seq =~ /AAAAA$/; # match AAAAA at end of $seq

• Notice that ^ at the beginning of a pattern means something different than ^ in a character class such as [^ACGT] or [^0-9]

• You can make sure that an entire string is matched by anchoring both the beginning and end of the match:

  • $seq =~ /^ATGCCCCAGCAGCAGCAGTTTTAAAAAA$/;
More new symbols in pattern matching

• There is a negative binding operator \(!~\) that means "does not match". Example:
  ```perl
  if ($seq !~ /A{10,}$/)  {
    # Did not find at least 10 A's at end of sequence
  }
  ```

• To match any character, use the symbol " . " (dot). To literally match a dot you must escape it as \.\.
  ```perl
  if ($seq =~ /^TT.TT/)  {
    # Begins with 2 T's, then any character, then 2 more T's
  } elsif ($seq =~ /\./) {
    # Has . at the end!
  }
  ```

• Parentheses \((\ ))\) can be used to group portions of a pattern. Example:
  ```perl
  if ($seq =~ /(CAG){30,}/)  {
    # Found 30 or more CAG's
  }
  ```
Capturing a Matched Pattern

• It is often useful to capture text that matches portions of a pattern

• Any segments of a pattern surrounded by parentheses ( ) are captured in special temporary variables

• These special variables are named $1, $2, etc.

• Example:

```perl
$pseq = "THISMAYBEAPROTEINSEQUENCE";
if ( $pseq =~ /([JOUX])/i ) {
    print "$1 is not a valid amino acid residue\n";
}
```

# How many characters are captured?
# Why do pattern capture variables not start with $0?
# Note that $1, $2, etc. will be overwritten by the next pattern search!
**in VIM**

- You can also use regex for replacement!
- `:%s/<search>/<replace>/g`
  - Uses `\(...\)` for groups
  - `\1, \2` for references
  - `\0` is whole match
in UNIX, `sed`

- Usage is similar to the search and replace in vim (actually it's exactly the same)
  ```
  sed "s/<search>/<replace>/[g]" <file>
  ```
# Special Characters in Regular Expressions

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Match any single character</td>
</tr>
<tr>
<td>^</td>
<td>Anchor match at beginning of string</td>
</tr>
<tr>
<td>$</td>
<td>Anchor match at end of string</td>
</tr>
<tr>
<td>?</td>
<td>Match preceding element 0 or 1 time</td>
</tr>
<tr>
<td>*</td>
<td>Match preceding element 0 or more times</td>
</tr>
<tr>
<td>+</td>
<td>Match preceding element 1 or more times</td>
</tr>
<tr>
<td>{n,m}</td>
<td>Match preceding element n to m times</td>
</tr>
<tr>
<td>[ ]</td>
<td>Match any character in character class</td>
</tr>
<tr>
<td>[^ ]</td>
<td>Match any character NOT in character class</td>
</tr>
<tr>
<td>( )</td>
<td>Group and capture expression</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>Escape the character immediately following \</td>
</tr>
</tbody>
</table>