2.1.5 Variable scoping

Throughout the previous section all the examples have used the syntax:

```
my $var = "value";
```

The `my` is actually not required; you could just use:

```
$var = "value";
```

However, the above usage will create global variables throughout your program, which is bad programming practice. `my` creates lexically scoped variables instead. The variables are scoped to the block (i.e. a bunch of statements surrounded by curly-braces) in which they are defined.

```
my $a = "foo";
if ($some_condition) {
    my $b = "bar";
    print $a;           # prints "foo"
    print $b;           # prints "bar"
}
print $a;               # prints "foo"
print $b;               # prints nothing; $b has fallen out of scope
```

Using `my` in combination with a `use strict;` at the top of your Perl scripts means that the interpreter will pick up certain common programming errors. For instance, in the example above, the final `print $b` would cause a compile-time error and prevent you from running the program. Using `strict` is highly recommended.

2.1.6 Conditional and looping constructs

Perl has most of the usual conditional and looping constructs except for case/switch (but if you really want it, there is a Switch module in Perl 5.8 and newer, and on CPAN. See the section on modules, below, for more information about modules and CPAN).

The conditions can be any Perl expression. See the list of operators in the next section for information on comparison and boolean logic operators, which are commonly used in conditional statements.

if

```
if ( condition ) {
    ...
} elsif ( other condition ) {
    ...
} else {
    ...
}
```

There's also a negated version of it:

```
unless ( condition ) {
    ...
}
```

This is provided as a more readable version of `if (!`*`condition`*`)`.

Note that the braces are required in Perl, even if you've only got one line in the block. However, there is a clever way of making your one-line conditional blocks more English like: