

Interpolation

Variable Interpolation,
Backslash Interpolation

Interpolation

- Sometimes called "substitution"
 - In Perl, "Substitution" means something else
- Replacing symbol/variable with its meaning/value within a string
- Two kinds of interpolation – variable and backslash
- Done *only* in double-quoted strings, not single-quoted strings.

Backslash interpolation

- aka: character interpolation, character escapes, escape sequences.
- When any of these sequences are found inside a double – quoted string, they're interpolated
- All escapes listed on page 61 of Camel
 - And in `perldoc perllop` (search for 'alarm')
- Most common: `"\n"`, `"\t"`

Backslashes in Reverse

- A backslash in a double-quoted string makes normal characters special.
 - makes 'n' into a newline, 't' into tab, etc
- Also makes special characters normal.
 - \$, @, \ are all special. If you want to use them in a double quoted string, must backslash them.
 - `print "My address is lallip@rpi.edu"`
 - Error, thinks @rpi is an array
 - use strict and warnings!!
 - `print "My address is lallip\@rpi.edu"`
 - Prints correctly.
 - preferred is to not use " if you only need '
 - `print 'My address is lallip@rpi.edu';`

Translation Escapes

- pg 61, table 2-2 of Camel
- `\u` – next character is uppercased
- `\l` – next character is lowercased
- `\U` – all characters until `\E` are uppercased
- `\L` – all characters until `\E` are lowercased
- `\E` – end `\U` or `\L`
- `my $name = 'paul';`
`print "Hi, my name is \u$name\n";`

Variable Interpolation

- variables found within " " are interpolated.
- ' ' strings are NOT searched for interpolation
- `my $foo = 'hello';`
- `my $bar = '$foo world';`
 - \$bar gets value: \$foo world
- `my $bar2 = "$foo world";`
 - \$bar2 gets value: hello world
 - Future changes to \$foo do NOT affect \$bar2!!!

Don't confuse the parser

- perl looks in double-quoted strings for anything that looks like a variable.
- The parser stops only when it gets to a character that cannot be part of the variable name
- `my $thing = 'bucket';
print "I have two $things\n";`
- perl assumes you are printing a variable \$things
 - always use `strict; !!`
- Specify where the variable ends with `{}`
`print "I have two ${thing}s\n";`
- Possibly unexpected: the `'` can be a valid part of a variable name.
 - Depreciated method of fully qualifying a package variable
 - `$main'foo` → `$main::foo`
 - `"I have $name's book"` → `$name::s`

What can interpolate?

- Scalars, arrays, slices of arrays, slices of hashes
 - NOT entire hashes
- Arrays (and slices) will print out each member separated by `$` (default: `' '`):
 - `my @array = (1, 3, 5, 7);`
 - `print "The numbers are @array.\n";`
 - outputs: The numbers are 1 3 5 7.
- Recall that printing an array (or list) directly is controlled by the `$,` variable
 - `print "The numbers are", @array, ".\n";`
 - The numbers are 1357.

Quote-like operators

- You might not always want to specify a string by double quotes:
 - `"He said, "John said, "blah""\n".`
 - You would have to backslash all those quotes
- Perl allows you to choose your own quoting delimiters, via the quote-like operators: `q()` and `qq()`
- A string in a `q()` block is treated as a single-quoted string.
- A string in a `qq()` block is treated as a double-quoted string.
