Day 14: Recipes III

Miscellaneous

Suggested Reading:
Perl Cookbook (2nd Ed.)

Chapter 1: Strings
Chapter 4: Arrays
Chapter 5: Hashes

Homework Review

Homework Preview

The Final Report

```
On July 29, the high was forecast to be in
the upper 80s and the actual high was 86.5F,
0.5F lower than predicted; the low was
forecast to be in the lower 60s and the
actual low was 71.2F, 8.2F higher than
predicted.

On July 30, the high was forecast to be in
the upper 80s and the actual high was 88.1F,
as predicted; the low was forecast to be in
the upper 60s and the actual low was 72.1F,
3.1F higher than predicted.
```

Miscellaneous Tricks

Swapping Values

- Common approach:

```
my $x = 12;
my $y = 30;
...
my $temp = $x;
$x = $y;
$y = $temp;
```

Parallel Assignment

- The Perl way

```
my $x = 12;
my $y = 30;
...
($x, $y) = ($y, $x);
```

- Another example: Fibonacci iteration

```
my $x = 0; my $y = 1;
while ($y <= $max) {
    ($x, $y) = ($y, $x + $y);
}
```

Making Sure a Variable is Defined

- The obvious way:

```
my $foo = get_value();  # can return undef
if (not defined $foo) {
    $foo = default value here;
}
```

- A common Perl way:

```
my $foo = get_value();  # can return undef
$foo ||= default value here;
```

- Any problems with the Perl way?

Long String Literals

Use special Perl syntax called a “here-document”

```
my $long_string = <<END;
This is the start of the string.
It can go on for many lines.
When done, terminate on the next line.
END

safe_write($filename, <<CONTENTS);
* File contents!
  - Formatting is preserved!!
* Interpolation even works: $foo!!!
CONTENTS
```

Really Long String Literals

- Use `__DATA__` section at end of script
- **Pro:** Part of script… **Con:** Part of script

```
# Perl opens DATA filehandle automatically
while (<DATA>) {
    # do stuff with lines here
}

__DATA__
This is the start of a very long data block.
It must be the last part of the script file.
Perl will not run code after __DATA__.
No $variable interpolation here.
```

Subroutines: Scalar, List, or Void Context?

```
sub read_file {
  my $filename = shift;

  # Read file contents into array
  open(my $filehandle, '<', $filename)
    or die "Could not open '$filename': $!\n";
  my @lines = <$filehandle>;
  close($filehandle);

  # Choose the correct return format
  return @lines if wantarray;
  return join('', @lines) if defined wantarray;
  return;
}
```

Subroutines: Scalar, List, or Void Context?

```
sub read_file { ... }  # from previous slide

my $scalar_contents = read_file('test.txt');
=> "foo\nbar\n42\n"

my @array_contents = read_file('test.txt');
=> ("foo\n", "bar\n", "42\n");

read_file('text.txt');
=> undef
```

Collection Tricks

Randomize Order of List

- Don't reinvent the wheel!
- Hard to implement correctly
- Use built-in **`List::Util::shuffle`**

```
use List::Util qw/shuffle/;

my @list = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
my @random_list = shuffle(@list);

=> [e.g.:] (7, 3, 2, 4, 1, 5, 8, 10, 6, 9)
```

- **`List::Util`** has other useful list functions: **`first`**, **`max`**, **`maxstr`**, **`min`**, **`minstr`**, **`reduce`**, **`sum`**

Fetch Hash Keys in Insertion Order

- Keep and manage separate, parallel array
- Deletions are painful

```
my %hash;
my @hash_keys;
foreach my $name (<$input_fh>) {
    unless (exists $hash{$name}) {
        $hash{$name} = $something;
        push(@hash_keys, $name);
    }
}

foreach my $key (@hash_keys) { … }
```

Hash as Set

Keys are set elements; values are always **1**:

```
map { $set{$_} => 1 } @elements_for_set;
```

Test for set membership:

```
if (exists $set{$element}) { … }
```

Compute *union* of two sets:

```
my %u = map { $_ => 1 } keys %set_a, %set_b;
```

Compute *intersection* of two sets:

```
my %i = map { $_ => 1 }
        grep(exists $set_b{$_}, keys %set_a);
```

Output Formatting

Large Numbers With Commas

```
sub commify {
  my $text = reverse $_[0];
  $text =~ s/(\d\d\d)(?=\d)(?!\d*\.)/$1,/g;
  return scalar reverse $text;
}

my $num_with_commas = commify(1234567.8901);

=> '1,234,567.8901'
```

- `(?=`*`pattern`*`)` : zero-width positive look-ahead
- `(?!`*`pattern`*`)` : zero-width negative look-ahead

Wrapped Text

- Don’t reinvent the wheel!

```
my $string = "This is a lot of text.  But it
is not very well formatted yet.  What can be
done about it?\n";

use Text::Wrap;          # Read perldoc for usage
$Text::Wrap::columns = 35;
print wrap('', '', $string);

This is a lot of text.  But it is
not very well formatted yet.  What
can be done about it?
```

Homework

The Final Report

```
On July 29, the high was forecast to be in
the upper 80s and the actual high was 86.5F,
0.5F lower than predicted; the low was
forecast to be in the lower 60s and the
actual low was 71.2F, 8.2F higher than
predicted.

On July 30, the high was forecast to be in
the upper 80s and the actual high was 88.1F,
as predicted; the low was forecast to be in
the upper 60s and the actual low was 72.1F,
3.1F higher than predicted.
```

Weather Analysis, Part III

- Compare forecasts to observations!
- What does "UPPER 80S" mean?
- Which days/hours to use for a given forecast?
  - Daily high, 4–6 p.m.; daily low, ~5 a.m. ***(of next day!)***
  - So use (*date*, 12 p.m.) – (*date* + 1, 11 a.m.)
- Display results in a paragraph of text
  - Format should follow sample
  - Sample may not show every case!
  - Wrap text to 72 columns (as if for email)