POD Translation by *pod2pdf*

ajf@afco.demon.co.uk

eated-Code-Is-a-Mistake--Appe

Table of Contents
Repeated-Code-Is-a-Mistake--Appendix

Bonus Slides

Identically Repeated Code

- It‘s usually easy to spot code which appears identically in more than one place.
- It probably got that way because you copied-and-pasted it.
- So placing that code in a subroutine is not difficult.

Similar ... But Not Repeated Exactly

- But I‘ve often found that I had to modify a section of code to get it to work properly in a new situation.
- How the shared aspects of the original and modified sections can be extracted and placed in a subroutine is not necessarily self-evident.
- Sometimes, you just have to stare at the code for a couple of months!

Mall::Tally: A Pseudo Spreadsheet

Another project from my day job.

A package named ***Mall::Tally*** takes data on the occurrence or non-occurrence of therapeutic groups in a psychiatric treatment program and displays them in spreadsheet-like text files.

```
 8   12    9   75   12    7   58   24   16   66
 9    7    6   85    8    2   25   15    8   53
10   16   10   62   17    7   41   33   17   51
11   12    8   66   13    5   38   25   13   52
12   14   10   71   10    0    0   24   10   41
14   13    8   61   10    6   60   23   14   60
     ... [other rows snipped] ...
98    0    0    0    1    1  100    1    1  100
     89   59   66   84   36   42  173   95   54
```

Column 1: Numerical code for ward or department.

Columns 2-4, 5-7: For each week: groups scheduled; groups which actually took place; percentage which took place.

Columns 8-10: For all weeks: total scheduled, occurred, percentage.

Bottom Row: Total scheduled/occurred/percentage across departments.

Code for Tallying Individual Groups

- Each treatment group has a unique ID ($_ below).
- Each group meets weekly unless excused. A group either occurs or fails to occur.
- Tallying is done in multi-week periods (typically, a month).

```
foreach my $period (@overall) {
  my %week = %{$period};
  foreach (keys %week) {
    my ($sch, $occ, $percent);
    $sch =     defined ($week{$_}{'sched'}) ? $week{$_}{'sched'} : 0;
    $occ =     defined ($week{$_}{'occur'}) ? $week{$_}{'occur'} : 0;
    $percent = $sch                         ? $occ / $sch * 100  : 0;
```

- This is part of the code to compute each row in the preceding slide.
- Note the data structure: an array (`@overall`) of hashes (`%{$period}`) of hashes (`%{$week{$_}}`). Note also the 3 uses of the ternary operator.

Code for Tallying the Bottom Line

The code for tallying the bottom line (totals across departments) was similar, but not identical, to the code for tallying individual groups.

```
foreach my $period (@overall) {
    my %week = %{$period};
    my ($sch, $occ, $percent);
    $sch =     defined ($week{'sched'}) ? $week{'sched'}    : 0;
    $occ =     defined ($week{'occur'}) ? $week{'occur'}    : 0;
    $percent = $sch                     ? $occ / $sch * 100 : 0;
```

Note the data structure: an array (`@overall`) of hashes (`%{$period}`) — one level shallower than the preceding. But we still use the ternary operator three times.

A Challenge

- These two passages of code occurred within the top and bottom parts of same subroutine.
- The sub was working; I was prepared to live with the repetitious code.
- But then I had to create a **new** subroutine which significantly altered other parts of the first subroutine but kept these parts the same.
- I faced the prospect of having **four** very similar blocks of code, rather than just two.
- I didn't want to be stymied by the difference between:

  ```
  $week{$_}{'scheduled'}
  ```

  and

  ```
  $week{'scheduled'}
  ```

After Staring at This for Several Months ...

I decided to focus just on the data for just one group for just one week: `%{$week{$_}}`. Its value is a hash which may have keys titled `'scheduled'`, `'occurred'`, and/or `'excused'`.
I pass a reference to that hash's value to a subroutine.

```
sub _calculate_group_week {
  my $v = shift;
  my %group_week = %{$v};
  my ($sch, $occ, $percent);
  $sch     = defined ($group_week{'sched'}) ? $group_week{'sched'} : 0;
  $occ     = defined ($group_week{'occur'}) ? $group_week{'occur'} : 0;
  $percent = $sch                           ? $occ / $sch * 100    : 0;
  return ($sch, $occ, $percent);
}
```

The top part of the original subroutine now looked like this:

```
foreach my $period (@overall) {
  my %week = %{$period};
  foreach (keys %week) {
    my ($sch, $occ, $percent) = _calculate_group_week(\%{$week{$_}});
```

For Each Top There Must Be a Bottom ...

I then revised the code passage for the bottom line in a similar manner:

```
sub _calculate_all_groups_week {
  my $v = shift;
  my %all_group_wk = %{$v};
  my ($sch, $occ, $percent);
  $sch     = defined ($all_group_wk{'sched'}) ? $all_group_wk{'sched'} : 0;
  $occ     = defined ($all_group_wk{'occur'}) ? $all_group_wk{'occur'} : 0;
  $percent = $sch                             ? $occ / $sch * 100      : 0;
  return ($sch, $occ, $percent);
```

```
}
```

The bottom part of the original subroutine now looked like this:

```
foreach my $period (@overall) {
  my ($sch, $occ, $percent) = _calculate_all_groups_week(\%{$period});
```

Eliminating Redundant Variables ... a Surprise!

Several of my variables were present only to enable me to penetrate the line noise endemic to Perl's multi-dimensional data structures.

```
my %week           = %{$period};
my %group_week     = %{$v};
my %all_group_week = %{$v};
```

Once I decided to eliminate these synthetic variables and put up with some line noise, I discovered that my two new subroutines were identical!

One Subroutine Replaces Two

```
sub _calculate_row {
  my $v = shift;
  my $sch     = defined (${$v}{'sched'}) ? ${$v}{'sched'}    : 0;
  my $occ     = defined (${$v}{'occur'}) ? ${$v}{'occur'}    : 0;
  my $percent = $sch                     ? $occ / $sch * 100 : 0;
  return ($sch, $occ, $percent);
}
```

It's the argument which I pass to the subroutine that makes the difference.

```
foreach my $period (@overall) {
  foreach (keys %{$period}) {
    my ($sch, $occ, $percent) = _calculate_row( \%{ ${$period}{$_} } );
```

versus

```
foreach my $period (@overall) {
    my ($sch, $occ, $percent) = _calculate_row( \%{$period} );
```

Have Someone Else Stare at Your Code

- Two months after writing the above code — and after I had spent a couple of weeks preparing this talk — I showed this code to a Perl expert who noted that in both instances of `_calculate_row()` I was dereferencing, and then taking a reference to, something that was already a hash reference.
- Top part before and after:

```
_calculate_row( \%{ ${$period}{$_} } );

_calculate_row( ${$period}{$_} );
```

- Bottom part before and after:

```
_calculate_row( \%{$period} );

_calculate_row( $period );
```

- Whose eagle eyes spotted this repeated code? Who else: Mark Jason Dominus.