Day 10: Correctness

Suggested Reading:

*Programming Perl* (3rd Ed.)
Chapter 20: The Perl Debugger

http://perldoc.perl.org/ — Modules
Test::Simple, Test::More, Test::Harness
Homework Review
Correctness
I wrote a script!
\o/

Did I do it right?
>_<
What Does “Do It Right” Mean?

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>functionality</td>
<td>Does it work correctly?</td>
</tr>
<tr>
<td>reliability</td>
<td>Does it work every time?</td>
</tr>
<tr>
<td>usability</td>
<td>Is it easy to use?</td>
</tr>
<tr>
<td>efficiency</td>
<td>Is it fast? Low memory, disk, I/O, …?</td>
</tr>
<tr>
<td>maintainability</td>
<td>Is it easy to change?</td>
</tr>
<tr>
<td>portability</td>
<td>Does it work well everywhere?</td>
</tr>
</tbody>
</table>

(adapted from ISO 9126-1)
### What About … Not So Right?

<table>
<thead>
<tr>
<th>Failure</th>
<th>Defect</th>
</tr>
</thead>
<tbody>
<tr>
<td>An <em>event</em>: Something went wrong; unexpected behavior</td>
<td>A <em>mistake</em>: Something is likely or certain to cause a failure</td>
</tr>
<tr>
<td>Hardware</td>
<td>Cracked solder joint</td>
</tr>
<tr>
<td>Network</td>
<td>Flaky router</td>
</tr>
<tr>
<td>Data</td>
<td>Data-entry errors</td>
</tr>
<tr>
<td>User</td>
<td>Hangover</td>
</tr>
<tr>
<td>Software</td>
<td>Bugs!</td>
</tr>
</tbody>
</table>
How to Test / Debug

• “Try it out”
• `print()` statements / logging
  – see, e.g., Log::Log4perl
• Debugger
• Automated testing
• Code review
• Formal analysis
Manual Testing
Manual Debugging: print()

```perl
my $DEBUG = 1;
sub convert {
    my ($from, $to, $value) = @_; 
    print "from = $from\n" if $DEBUG;
    print "to = $to\n" if $DEBUG;
    my $meters = $value * $UNITS{$_from};
    print "meters = $meters\n" if $DEBUG;
    my $result = $meters / $UNITS{$_to};
    print "result = $result\n" if $DEBUG;
    return $result;
}
```
Debuggers

Common debugger features:

- View code
- Step through (running) code, line-by-line
- Examine variables
- Run and stop at breakpoints
- Stack traces
- Watch points
# The Perl Debugger

**List/search source lines:**
- `l [ln|sub]` List source code
- `- or .` List previous/current line
- `v [line]` View around line
- `f filename` View source in file
- `/pattern/ ?patt?` Search forward/backward

**Control script execution:**
- `T` Stack trace
- `s [expr]` Single step [in expr]
- `n [expr]` Next, steps over subs
- `<CR/Enter>` Repeat last n or s
- `r` Return from subroutine
- `c [ln|sub]` Continue until position

**Debugger controls:**
- `o [...]

**Data Examination:**
- `x|m expr` Evals expr in list context, dumps the result or lists methods.
- `p expr` Print expression (uses script's current package).
- `S [!!]pat` List subroutine names [not] matching pattern
- `V [Pk [Vars]]` List Variables in Package. Vars can be ~pattern or !pattern.
- `X [Vars]` Same as "V currentPackage [Vars]". i.e class inheritance tree.
- `y [n [Vars]]` List lexicals in higher scope `<n>`. Vars same as V.
- `e` Display thread id

For more help, type `h cmd_letter`, or run `man perldebug` for all docs.
Automated Testing
Automated Testing

• Write software to test (other) software

• Humans vs. machines

• Types of automated tests
  – Unit tests
  – Functional tests
  – Performance tests
Test::Simple

Run a .t file to test a module

```perl
#!/usr/bin/perl
use strict; use warnings;

use SomeModule; # to be tested

use Test::Simple tests => 2;

ok(is_doing_ok(), 'doing ok');
ok(the_result() == 7, 'value ok');
```
use Test::More tests => 6;

ok(is_ok(), 'is OK');
is($the_answer, 42, 'answer ok');
isnt(exit_status(), 1, 'syscall');
like($name, qr/tim/i, 'good name');
unlike($result, qr/error/i, 'test');
diag("current value of name = $name");

SKIP: {
    skip 'not available', 1 if ...;
    ok(...);
};
Testing a Standalone Script

```perl
use Getopt::Long;
GetOptions('test' => \&run_tests);

# main script & subroutines here

sub run_tests {
    require Test::More;
    Test::More->import;
    plan(tests => nnn);
    # test subroutines here
    exit 0;
}
```
Unit Testing Tips

• Test logical chunks of code — usually subroutines
• Aim for reasonable coverage
• Run often!
  – After every (significant) change
  – Before you use, hand in, commit, …
• Encode past failures as tests
Test-First Development

• Radical idea: Write tests **FIRST**

• Then write code until tests pass
• Clarifies and documents design
• And of course… is useful for testing!

http://junit.sourceforge.net/doc/testinfeected/testing.htm
Wrap Up
Other Scripting Languages

- “Try it out” and printing/logging always work
- Most have debuggers and/or interactive modes
- Unit testing:
  - Most others are based on jUnit
  - Expect similar and richer assertions
  - Introspection rocks!
Homework

• Write a couple of simple but non-trivial functions
• Write unit tests against them
• Make script work in “test” and normal modes