Day 5: Subroutines


Chapter 4, Subroutines
Turn In Homework
Homework Review

• Hand in:
  – Just one single sheet of paper, ideally
  – Just your code, not its output

• Perl is not C: declare/define variables when used

• Beware repeated code…
Theory
Procedural Programming

- group statements into named units
- *aka* procedures, functions, subroutines, methods
- executed from anywhere, including self
- usually supports inputs and output(s)
But why?
E. W. Dijkstra (1968)
Go to statement considered harmful

*Communications of the ACM, 11, 147–148*
Don’t Repeat Yourself (DRY)

- Code
- Data
- Configuration
- Documentation

Hunt & Thomas (1999), *The Pragmatic Programmer*
OK, OK, but... when?
Make a Subroutine...

- For repeated code (DRY)

- For logical organization
  - capture main flow vs. parts
  - break up excessively long parts
  - scope control

- For testing

- Already been using: `print`, `chomp`, `open`, ...
Theory Code
Defining a Subroutine

sub subroutine_name {
    # statements go here
}

• Put nearly anywhere (except in another `sub`)
• Namespace is distinct (but don’t abuse this!)
Using a Subroutine

&subroutine_name;
subroutine_name();
&subroutine_name();

• &: almost always OK, often optional
• (): often optional, sometimes helpful

• A subroutine call is an expression:

&halt_cpu if temperature_too_high();
Arguments

remember automatic variables?
within the subroutine, arguments are in @_

sub be_silly {
    if ($_[0] > $_[1]) { ... }
    my $named_argument = $_[2];
    foreach my $argument (@_) { ... }
}
Better Arguments

sub idiom_1 {
    my ($foo, $bar) = @_;
    ...
}

sub idiom_2 {
    my $foo = shift;  # @__ is implied
    my $bar = shift;  # @__ is implied
    ...
}
Return Values

Default: Return the last expression *evaluated*

```perl
sub bigger {
    my ($a, $b) = @_;  # Assign values to $a and $b
    if ($a > $b) { $a } else { $b }
}
```

Often clearer: An explicit `return`

```perl
return;
return 42;
return "Hello, $name\n";
```
Perl lets you return lists, too:

```perl
sub how_do_i_love_thee {
    ...
    return @the_ways;
}

my @array = how_do_i_love_thee();
print "Let me count... "
print scalar(@array);
print "\n";
```
Scoping

my $answer = 42;
print "The answer: $answer\n";

contemplate($answer);

sub contemplate {
    my $answer = shift;
    $answer /= 6;
    print "Upon further review: $answer\n";
}
ask_questions()
Other Scripting Languages

- All have procedures
- Syntax varies widely
- Look for:
  - explicit declaration of argument signature
    [Ruby] `def foo(arg1, arg2, blah)`
  - default arguments
    [Ruby] `def bar(arg1, arg2 = 42)`
  - different scoping rules (PHP)
  - different requirements (Python requires return)
  - anonymous functions / closures (Ruby)
Homework

• Unit conversions!
• Subroutine usage is a bit forced…
• Think about (internal) data and DRY