Day 5: Subroutines

suggested reading:

Learning Perl (4th Ed.),
Chapter 4: Subroutines

TURN IN HOMEWORK

HOMEWORK REVIEW

Today's Problem

- Script accepts inputs from user
- Must validate inputs
- Input taken at different places
- Don't want to repeat ourselves

Subroutines

- Reuse code many times in 1 script
- Central to procedural programming
- Sometimes required (callouts)

- Already used: print, chomp, open, …

Don't Repeat Yourself (DRY)

- Code (OAOO)
- Data
- Configuration
- Documentation

Hunt & Thomas, *The Pragmatic Programmer*

When should you write a subroutine?

Write a Subroutine…

- For repeated code (OAOO/DRY)
- For logical organization
  - capture main flow vs. parts
  - break up excessively long sections
- For testing

Defining a Subroutine

```
sub subroutine_name {
  # code goes here
}
```

- Put just about anywhere (except within another subroutine)
- Namespace is distinct from variables (but don't abuse this)

Using a Subroutine

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

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

- Can use in expressions:

```
&wear_jacket if it_is_cold();
```

Arguments

```
compute_question($universe, 42);
```

- within the sub, arguments are in **@_**

```
sub do_stuff {
  if ($_[0] > $_[1]) { ... }
  my $named_argument = $_[2];
  foreach my $x (@_) { ... }
}
```

Better Argument Idioms

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

```
sub idiom_2 {
  my $foo = shift;    # @_ is implied
  my $bar = shift;    # @_ is implied
  ...
}
```

Return Values

- Last expression evaluated

```
sub bigger {
  my ($a, $b) = @_;
  if ($a > $b) { $a } else { $b }
}
```

- **return** operator

```
return;
return 42;
return $foo;
```

Returning Lists

- You can return a list, too

```
sub foo {
  # blah
  return @list;
}

my @results = foo();
my ($a, $b, $c) = foo();
```

Scoping

```
my $foo = 42;
do_something($foo);

sub do_something {
  my $foo = shift;
  $foo += 8;
  print "$foo\n";
}
print "$foo\n";
```

Other Scripting Languages

- syntax varies widely
- function signatures with arguments
  - [Ruby] **`def foo(arg1, arg2, blah)`**
- default arguments
  - [Ruby] **`def bar(arg1, arg2 = 42)`**
- Python: **`return`** required, **`pass`**
- PHP: import globals explicitly
- Ruby: blocks as function arguments