Day 6: References

suggested reading:

perlreftut

http://perldoc.perl.org/perlreftut.html

or "perldoc perlreftut"

Turn In Homework

Yesterday's Homework

What we have

- $ A scalar
  - Holds one thing.
- @ An array
  - Holds a bunch of scalars.
- % A hash
  - Holds a bunch of scalars.
- What if I need something more complex?

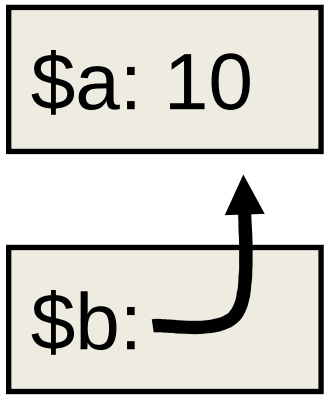
References

(Similar to pointers, for you C and Pascal fans)

References

- A reference is a scalar that refers to some other variable

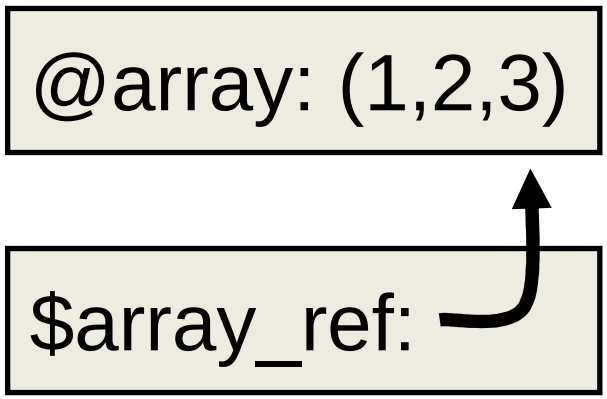
```
my $a = 10;
my $b = \$a;
${$b} = 5;
print $a; # prints "5"
```



Creating references

- You can point a scalar at a named variable:

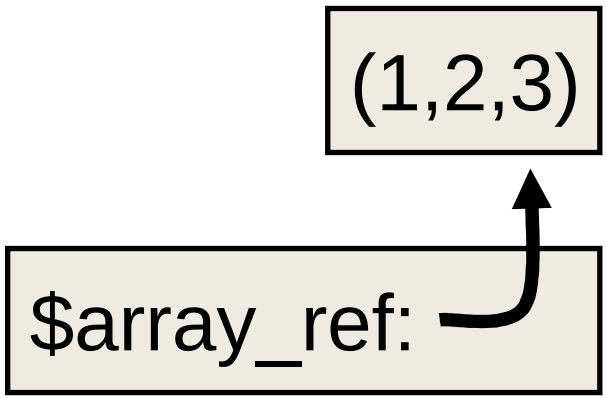
```
my $array_ref  = \@array;
my $hash_ref   = \%hash;
my $scalar_ref = \$scalar;
```



Creating references

- You can point a scalar at an anonymous array or hash:

```
my $array_ref = [ 1, "two", 3.0 ];
my $hash_ref = { "one" => 1,
                 "two" => 2 };
```



Using references

- Put `${}` around a scalar ref to look inside.

```
my $scalar_ref = \$scalar;
```

  - `$scalar` and `${$scalar_ref}` are identical

```
$scalar = 1;
${$scalar_def} = 2;
print "$scalar == 2\n";
```

Using references

- `@{}` and `%{}` work the same way. Remember to use $ to look at individual entries!

```
my $array_ref = \@array;
```

- `@array` and `@{$array_ref}` are identical
- `$array[2]` and `${$array_ref}[2]` are identical

Using references

```
my $hash_ref  = \%hash;
```

- `%hash` and `%{$hash_ref}` are identical
- `$hash{'test'}` and `${$hash_ref}{'test'}` are identical

Abbreviations: ->

- These are identical
  - `$array[2]`
  - `${$array_ref}[2]`
  - `$array_ref->[2]`
- These are identical
  - `$hash{'test'}`
  - `${$hash_ref}{'test'}`
  - `$hash_ref->{'test'}`

Abbreviations: [1][2] {1}{2}

- You can omit the -> between indices
- These are identical
  - `${${$x[1]}{'fred'}}[9]`
  - `$x[1]->{'fred'}->[9]`
  - `$x[1]{'fred'}[9]`

Nesting

- You can nest these forever.
  - Or until you run out of memory.
  - Whichever comes first.

Uses

Grades (Hash of arrays)

- There are 5 homework assignments, each is worth 0, 1, or 2 points.
- Here are my grades:

```
my(@grades) = (0,1,1,2,2);
```

- There are multiple students.

```
my(%class) = (
    'Alan' => \@grades,
    'Nick' => [2,1,2,2,2],
    'Tim' => [2,2,2,2,2],
);
```

Let's run the averages

```
foreach my $student (key %class) {
   my $grade_ref = $class{$student};
   my @grades = @{$grade_ref};
   my $total = 0;
   foreach my $score (@grades) {
      $total += $score;
   }
   my $average = $total / scalar(@grades);
   print "$student: $average\n";
}
```

Bowling (Array of arrays)

- Bowling is broken into 10 frames, each with 1, 2, or occasionally 3 balls. You score for each ball.
- Easily held in an array of arrays!
- my(@scores) = ([5,0], [7,1], [10], [0,0], [3,0], [4,0], [0,4], [3,4], [10], [0,0]);

Bowling (Hash of arrays of arrays)

```
$bowling{'Alan'}[1][0] = 5;
$bowling{'Alan'}[1][1] = 0;
$bowling{'Alan'}[2][0] = 7;
$bowling{'Alan'}[2][1] = 3;
my(%bowling) = (
    'Alan' => [ [5,0], [7,3] ],
    'Joe' => [ [10,0], [10,0] ],
    'Chris' => [ [8,1], [8,1] ],
);
```

Structured data

- Perl has no direct equivalent to a class or struct in Java/C++
- Use hashes (of hashes) to store structured data

```
$movies{'Brazil'}{'Director'} =
  'Terry Gilliam';
$movies{'Brazil'}{'Actors'}[0] =
  'Jonathan Pryce';
```

- Perl's object system built on this

More complex data structures

- Linked lists
  - Or you can use an array
- Trees
  - Or you can use a hash
- Directed graphs
  - You could use a hash

Linked List Example

Really, just use an array

Passing into functions

- How to pass two arrays to a single function?

```
@a1 = (1,2);
@a2 = (3,4);
myfunc(@a1, @a2);
```

- Turns into (1,2,3,4)!

Passing into functions

```
myfunc(\@a1, \@a2);
sub myfunc {
  my($a1_ref, $a2_ref) = @_;
  my @a1 = @{$a1_ref};
  my @a2 = @{$a2_ref};
}
```

Passing out of functions

- What if you want to modify the thing passed in. (chomp!)

```
$string = "test\n";
my_chomp(\$string);
sub my_chomp {
  if(substr(${$_[0]}, -1, 1) eq "\n")
  {
    ${$_[0]} = substr(${$_[0]}, 0, -
  1);
  }
}
```

Other languages

- Structured data via class/object
  - Ruby, Python, Javascript, etc.
  - Typically: variable.member_variable

Python

- Nested data structures Just Work

```
# Array holding an array
a = [1, 2, ['3.1', '3.2]', 4]
# a[2][0]  is '3.1'
# Hash holding an array
d = {'a' : 'b', 'c' : [1,2] }
# d['c'][1] is 2
```

Ruby

- Nested data structures Just Work

```
# Array holding an array
a = [1, 2, ['3.1', '3.2]', 4]
# a[2][0]  is '3.1'
# Hash holding an array
d = {'a'=> 'b', 'c'=> [1,2] }
# d['c'][1] is 2
```

Ruby and Python references

- Like Java, variables for objects are references
- Hashes and arrays are objects

```
a = [1, 2]
b = a
b[0] = 'fred' # Modified a[0]!
```

So how do you copy arrays and hashes in Ruby and Python?

- Python:

```
array_b = list(array_a)
hash_b = hash_a.copy()
```

- Ruby

```
array_b = array_a.clone()
hash_b = hash_b.clone()
```

- These are shallow copies!
  - Sub arrays and hashes are still shared!

Some Philosophy

The scripting mindset

- Your time versus the computer's
- Computers are fast
  - Moore's law says next month's computer is 10% faster*

    * Moore's law says nothing of the sort
- ***When in doubt, waste the computer's time, not yours***
- Scripting languages based around this idea

Clarity, Correctness, and Efficiency

- Sometimes you make tradeoffs
- The tradeoffs vary based on the task
- A good general rule
  1. Clarity
  2. Correct
  3. Efficient

…premature optimization is the root of all evil.

- Donald Knuth (paraphrased)