Day 3: Collections

Suggested reading: 
*Learning Perl* (4th Ed.)
Chapter 3: Lists and Arrays
Chapter 6: Hashes
Turn In Homework
Homework Review
Write code.
At least a little.
Every day.
$: Scalar Variables

• $ prefix means scalar
• Holds one value
• Usually a number or string
• What if we want to collect values?
@: Array

@ prefix means *array* (aka *list*, *sequence*, *tuple*)

Ordered collection of scalar elements

Can mix element types in one list

0–n elements, bounded by memory
Making Arrays

• array literal syntax: (... , ... , ...)
• can assign lists ... even to a list of scalars
• beware of flattening

```perl
my @bike_gear = ('helmet', 'lock');
my @stuff = ('backpack', @bike_gear);

@stuff => ('backpack', 'helmet', 'lock')

my ($first, $second) = @stuff;
my ($start, @rest) = @stuff;
my ($one, $two, $three) = @bike_gear;
```
Using Arrays

- prefix with @
- on first use, declare with my
- reference an element with [...] (and $)
- element indexes start at 0

```perl
my @array;
$array[0] = 'CS 301';
$array[1] = 'CS 367';
print "First class: $array[0]\n";
$array[1] .= ' (data structures)';
print "Whole array: @array\n";
```
@array
$array[n]
Array Bounds

• arrays grow to fit maximum index
• limited only by memory
• accessing new or unassigned index => undef

my @array;
defined($array[0]);  => undef
$array[42] = 'The Answer';
defined($array[41]); => undef
defined($array[42]); => 1
defined($array[43]); => undef
Useful Array Operations

my @stack = (1, 2, 3);
my $top = pop @stack;       # @stack = (1, 2)
push @stack, 4;             # @stack = (1, 2, 4)

my @queue = (1, 2, 3);
my $next = shift @queue;    # (2, 3)
push @queue, 4;             # (2, 3, 4)
unshift @queue, 5;          # (5, 2, 3, 4)

print join( ' : ' , @queue) . "\n";
# => "5 : 2 : 3 : 4\n"
%
%: Hash

% prefix means hash (aka hash table, map, dictionary)

Arbitrary association from string key to scalar

Scalar values can be any type

0–n key-value pairs, bounded by memory

access is O(1), on average
Making Hash(es)

- literal: (key₁ => value₁, key₂ => value₂, ... )
- can use array literals (taken in pairs) — confusing!
- caution: order of keys is lost

```perl
my %map = (  
    '1289'     => 'CS 368, Section 1',  
    4265      => "Tim's office",  
    'CS 1240' => 'fancy lecture hall'  
);  
```
Using Hashes

• prefix with `%`
• on first use, declare with `my`
• reference an element with `{…}` (and `$`)
• keys are unique

```perl
my %hash = ('2001' => 'Arthur Clarke');
print "2001's author: $hash{'2001'}\n";
$hash{'I, Robot'} = 'Issac Asimov';

my %tallies;
$tallies{'foo'} = 1;
++$tallies{'foo'};
```
$hash \{ key \}$
Useful Hash Operations

see if a key exists

if (exists($my_hash{'ThisKey'})) { ... }

delete a key-value pair

delete $my_hash{'Goner'}

going get all keys (as array in arbitrary order)

my @key_list = keys %my_hash
Hashes as Sets

• set all values to (e.g.) 1
• easy and fast to check set membership
• great for finding unique things

```perl
my %seen;  # set of observed names
foreach my $name (@names) {
    $seen{$name} = 1;
}

if ($seen{$new_name}) {
    print "I have already seen $new_name\n";
}
```
$\texttt{foo } \neq \ @\texttt{foo } \neq \ %\texttt{foo}$
Size of Array or Hash

use **scalar** for array size

\[
\text{my } \$\text{size} = \text{scalar}(@\text{array});
\]

for hash, **keys** is array…

\[
\text{my } \$\text{size} = \text{scalar}(\text{keys } \%\text{hash});
\]

**length** is *only* for strings

\[
\begin{align*}
\text{my } \$\text{len} & = \text{length}($\text{some_string}); \quad \# \text{ OK} \\
\text{my } \$\text{len} & = \text{length}(@\text{array}); \quad \# \text{ horribly bad} \\
\text{my } \$\text{len} & = \text{length}(\%\text{hash}); \quad \# \text{ horribly bad}
\end{align*}
\]
Loops

```perl
for (my $i = 0; $i < scalar(@x); $i++) {
    print "element $i = $x[$i]\n";
}

foreach my $e (@x) {
    print "$e\n";
}

foreach my $key (keys %hash) {
    print "$key => $hash{$key}";
}

while (my ($key, $value) = each %hash) {...}
```
Phew!
Other Scripting Languages

• All have arrays and associative arrays

• Check for different or additional:
  – Terminology (list, map, dictionary, …)
  – Syntax ([ ] vs. { }, \texttt{len(array)} vs. \texttt{array.length})
  – Operations (sort, unique elements, flatten, shuffle)
  – Collections (e.g., set)
Homework

• Implement a primitive Metacritic-like system
  – Collect reviewer names and scores
  – Report scores and average score

• BE SURE TO LABEL YOUR PRINTOUT!!!

#!/usr/bin/perl

# Homework for CS 368-1
# Assigned on Day 03, 2010-07-15
# Written by Your Name Here

use strict;
use warnings;