Day 3: Collections

Suggested reading: *Learning Perl* (6th Ed.)

Chapter 3: Lists and Arrays

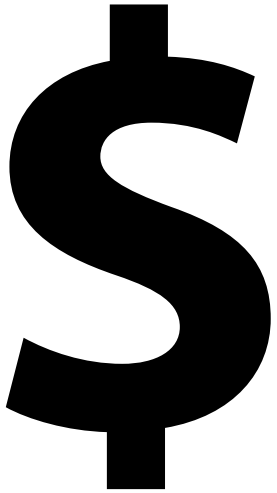
Chapter 6: Hashes

Turn In Homework

Homework Review

Will not be posted online

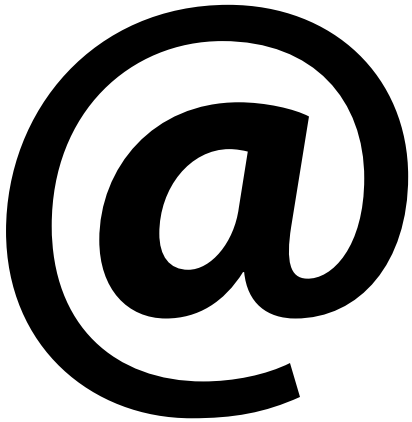
Write code.
At least a little.
Every day.
Have fun!



$: Scalar Variable

- **$** prefix means ***S****calar*
- Holds one value
- Number or string *(for now)*

How can we have a *collection* of (related) values?



@: Array

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

Ordered collection of scalar elements

0–*n* elements, limited only by memory

42	3.141	Tim	Hello	0

Making Arrays

- Array literal syntax: (…, …, …)
- Can assign lists … even to a list of scalars
- Beware of flattening

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


```
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**

```
my @array;
$array[0] = 'CS 301';
$array[1] = 'CS 367';
```

@array:

Index:	0	1
Contents:	CS 301	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);          # (1, 2, 3)
my $top = pop @stack;           # (1, 2)
push @stack, 4;                 # (1, 2, 4)

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

push @queue, 4;                 # (5, 2, 3, 4)

join(' : ', @queue)    # => '5 : 2 : 3 : 4'
```

%%

%: Hash

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

Unordered pairing from ***string*** key to scalar value

0–*n* key-value pairs, limited only by memory

Access is fast (***O***(1) on average)

Making Hashes

```
my %map = (
    'Eng 3024'  => 'Lecture Hall',
    4265        => "Tim's office",
    1240        => 'OSG School'
);
```

Key	*Value*
4265	Tim's office
1240	OSG School
Eng 3024	Lecture Hall

- Order of keys may change
- Alternate syntax uses pairs of elements in a list — too confusing!

Using Hashes

- Prefix with **%**
- On first use, declare with **my**
- Reference an element with **{…}** (and **$**)
- Keys are unique

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

my %count;
$count{'foo'} += 1;  # undef converts to 0
print $count{'foo'} if $count{'foo'} > 0;
```

```
%hash
```

```
$hash{key}
```

Useful Hash Operations

See if a key exists

```
if (exists($my_hash{'ThisKey'}) { … }
```

Delete a key-value pair

```
delete $my_hash{'Goner'}
```

Get all keys (as array, in arbitrary order)

```
my @key_list = keys %my_hash
```

Hashes as Sets

- Always assign values to **1** (e.g.)
- Easy and fast to check set membership
- Great for finding unique things

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

if ($seen{'Tim'}) {
  print "I have already seen Tim\n";
}
```

```
$foo ≠ @foo ≠ %foo
```

Size of Array or Hash

Use **`scalar`** for array size

```
my $size = scalar(@array);
```

For hash, **`keys`** gives an array, so…

```
my $size = scalar(keys %hash);
```

`length` is ***only*** for strings

```
my $len = length($some_string);          # OK
my $len = length(@array);    # horribly bad
my $len = length(%hash);     # horribly bad
```

Looping Over Arrays

The C way:

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

The Perl way:

```
foreach my $element (@some_array) {
    print "$element\n";
    # More statements are fine
    # Can use next and last here
}
```

Looping Over Hashes

Use the list of keys, one at a time, to access values:

```
foreach my $key (keys %some_hash) {
    print "$key => $some_hash{$key}\n";
}
```

Access keys and values, one pair at a time:

```
while (my ($key, $value) = each %hash) {
    print "$key => $value\n";
}
```

Selecting a Collection

Tips on Selecting a Good Collection

- Use a ***list*** when…
  - Have one datum per element
  - Order matters (e.g., deck of cards, events)
  - Access elements in sequence
  - Access elements by sequential number (the index)
- Use a ***hash*** when…
  - Have paired data (e.g., key/value) per element
  - Access elements haphazardly (by string key)
    - Scanning a list for one element is ***slow***
    - Looking up a value by its key in a hash is ***fast***
  - Instead of parallel arrays
  - Need something like a set

Phew!

Other Scripting Languages

- All have arrays and associative arrays
- Check for different or additional:
  - **Terminology** (list, map, dictionary, …)
  - **Syntax** (`[]` vs. `{}`, `len(array)` vs. `array.length`)
  - **Operations** (sort, unique elements, flatten, shuffle)
  - **Collections** (e.g., set)

Homework

- Implement a primitive grocery list
  - Collect grocery items and their prices
  - Report items and total cost
- BE SURE TO LABEL YOUR PRINTOUT!!!

```
#!/usr/bin/perl

# Homework for CS 368-3
# Assigned on Day 03, 2012-06-25
# Written by Your Name Here

use strict;
use warnings;
```