

Day 3: Collections

Suggested reading:

Learning Perl (4th Ed.)

Chapter 3: Lists and Arrays

Chapter 6: Hashes

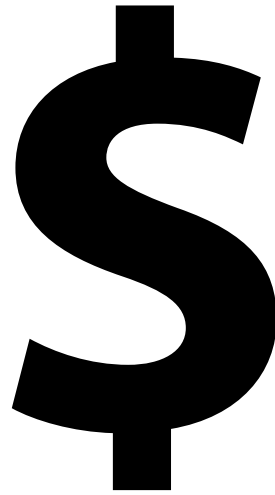
Turn In Homework

Course Prerequisite

Homework Review

Will not be posted online

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

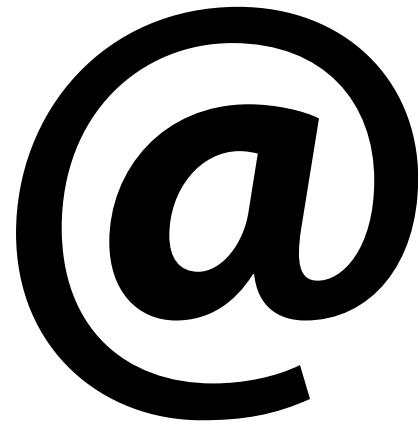


\$: Scalar Variable

- \$ prefix means *Scalar*
- 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);
```

@stuff =>

<i>backpack</i>	<i>helmet</i>	<i>lock</i>
-----------------	---------------	-------------

```
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 = (  
    'Psych 103' => 'Lecture Hall',  
    4265       => "Tim's office",  
    1240       => 'Lecture Hall'  
);
```

<i>Key</i>	<i>Value</i>
4265	Tim's office
1240	Lecture Hall
Psych 103	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

- set all values to (e.g.) 1
- 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
```


Loops

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

```
foreach my $element (@some_array) {  
    print "$element\n";  
}
```

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

```
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. `{}`, `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 368-1
```

```
# Assigned on Day 03, 2011-07-14
```

```
# Written by Your Name Here
```

```
use strict;  
use warnings;
```