# Day 3: Collections

suggested reading:

Learning Perl (4th Ed.),

Chapter 3: Lists and Arrays
Chapter 6: Hashes

# $

Scalar values

# $ Single, or scalar, values

- `my $bender = "robot";`
- `my $answer = 42;`
- `my $fred = undef;`

# @

Arrays

AKA

Lists, Sequences, Tuples

# @ Arrays

```
my @array_name =
  (scalar_1, scalar_2,…, scalar_n);
```

- For example…

```
my @futurama = ("Bender", "Fry",
  "Fry", "Leela");
```

# @ Arrays

```
my @futurama = ("Bender", "Fry",
  "Fry", "Leela");
```

- To address a single item, you use $
  - `$futurama[0]` is "Bender"
  - `$futurama[1]` is "Fry"

# @ Arrays

- Easy:
  - Read a numbered place

```
print $futurama[1];
```

  - Write a numbered place

```
$futurama[2] = "Zoidberg";
```

- Hard
  - Find a particular value
    - Where is "Fry"?

# @ Arrays

- Useful for ordered information

```
my @US_Presidents = ("Washington",
  "Adams", "Jefferson");


my @days_of_week = ("Sun", "Mon",
  "Tue", "Wed", "Thu", "Fri",
  "Sat");
```

# @ Arrays

```
my @array;
$array[10] = "Ten!";
```

- `$array[0]` through `$array[9]` automatically exist, but are undefined

# @ Arrays

- You can easily add and subtract items; array can resize as needed
  - `push, pop, shift, unshift` add and remove from beginning and end of array
  - `delete` can delete from the middle
  - Array slicing returns or modifies subsets

# Array flattening

- Perl flattens arrays

```
my @x = (1, 2);
my @y = (9, @x, 9, @x);
```

- This is equivalent to

```
my @y = (9, 1, 2, 9, 1, 2);
```

# Array assignment

```
my @array = (1, 2, 3);
my($one, $two) = @array;
```

- Now `$one` is 1 and `$two` is 2
- The 3 was just ignored

```
my($a, @b) = @array;
```

- Now `$a` is 1, `$b[0]` is 2, and `$b[1]` is 3. `@b` slurped up the rest

# Printing arrays

- `my(@array) = ('a', 'b', 'c');`
- Print second element:
  - `print "Second: $array[1]";`
  - Output is `"Second: b"`

- Print entire queue:
  - `print "Array: @array";`
  - Output is `"Array: a b c"`

# Joining arrays

- You can use loop to print an array nicely, but there is a useful shortcut:

```
my(@array) = ('a', 'b', 'c');
my $string = join(', ', @array);
print "Array: $string.";
```

- Output: `Array: a, b, c.`

# @ Arrays as ordered information

- Baseball scores

```
my @scores;
for (my $i = 0; $i < 9; $i++) {
  my $in = $i + 1;
  print "Inning $in score?\n";
  chomp(my $score = <STDIN>);
  $scores[$i] = $score;
  print "Scores: @scores\n";
}
```

# @ Arrays as queues

- Movies in your Netflix queue

```
my @netflix_q = ("12 Monkeys", "Time
  Bandits", "Brazil");
my $next_dvd = shift @netflix_q;
```

  - $next_dvd is now "12 Monkeys"
  - @netflix_q is now ("Time Bandits", "Brazil");

```
push @netflix_q, "Munchausen";
```

  - @netflix_q is now ("Time Bandits", "Brazil", "Munchausen");

# @ Arrays as stacks

```
my @commands = ('select', 'bold',
  'delete');
my $undo = pop @commands;
# $undo is now 'delete'
# @commands is now
#         ('select', 'bold')

push @commands, 'italics';
# @commands is now ('select',
  'italics')
```

# %

Hashes

# % Hashes

- AKA dictionaries, associative arrays, maps

```
my %authors = (
   "Dark Tower"   => "Stephen King",
   "Harry Potter" => "J.K. Rowling",
   "Discworld"    => "T. Pratchett",
   "Johnny"       => "T. Pratchett",
);
```

Key

Value

# % Hashes

```
my %authors = (
  "Dark Tower"   => "Stephen King",
  "Harry Potter" => "J.K. Rowling",
  "Discworld"    => "T. Pratchett",
  "Johnny"       => "T. Pratchett",
);
```

- Relates scalars to scalars.
- Keys must be unique

# % Hashes

```
my %authors = (
  "Harry Potter" => "J.K. Rowling",
  "Discworld"    => "T. Pratchett",
);
```

- "Who wrote Discworld?"
  - Easy: `print $authors{"Discworld"};`
- "Conan was written by Howard."
  - Easy: `$authors{"Conan"} = "Robert Howard";`

# % Hashes

```
my %authors = (
  "Harry Potter" => "J.K. Rowling",
  "Discworld"    => "T. Pratchett",
);
```

- "What did Pratchett write?"
  - Hard: walk the hash looking

# % Hashes

```
my %authors = (
  "Harry Potter" => "J.K. Rowling",
  "Discworld"    => "T. Pratchett",
);
```

- "=>" is (mostly) identical to ","
- Hash into an array is just the pairs
- Array into a hash assumes key, value, key, value, etc

# % Hashes

- No inherent order to the keys
  - Assume they come back in the worst possible order!
- Useful for associating values to other values.
  - A series with the author.
  - A word with its definition.
  - A username with a password.

# Login system

- A login system:

```
my %passwords = (
   'root' => 'k8H6h%4A',
   'bob' => 'secretcode!');
```

- Is the user name valid?
  - `exists($passwords{$username})`
- Is the password valid?
  - `$passwords{$username} eq $pass`

# Login system

- Add a user
  - `$passwords{$newuser} = $newpass;`
- Remove a user
  - `delete $passwords{$olduser};`

# % Hashes as sets

- Can use as a set. Useful for "is this part of the set" questions. Spam filtering:

```
my %spammers = ('malware@example.com' => 1,
                'scammer@example.org' => 1);
if(exists($spammers{$email}) ) {
    print "Refuse email from $email: SPAM\n";
}

# Variant test:
if( $spammers{$email} ) {
```

# % Hashes as sets

- Sets are useful for tracking things seen.

```
my %seen;
foreach my $email (@emails) {
    $seen{$email} = 1;
}
print join(", ", keys(%seen));
```

- (Shorter forms exist)

# $ @ %

- `$foo`, `@foo`, and `%foo` are three different variables.
  - Different namespaces.
- `$foo[1]` is the second element of `@foo`
- `$foo{1}` is an element of `%foo`

# How big is my array?

- If you try to use an array where only a scalar makes sense, Perl will return the size of the array
  - `my $size = @array;`
  - or more explicitly…
  - `my $size = scalar(@array);`
  - Very Perl specific!

# How big is my hash?

- "`scalar %hash`" doesn't work
- You can use "keys" to get an array of of the indices for the hash.
  - `my $size = scalar(keys(%hash));`
  - `my $size = keys(%hash);`

# length

- RIGHT: `length("some string")`
  - `(It's 11)`

- WRONG: `length(@foo)`
- WRONG: `length(%foo)`

# Looping over collections

# Loops: foreach

```
# Obviously this works
for (my $i = 0; $i < @x; $i++) {
    print "$x[$i]\n";
}
# Sometimes more handy:
foreach my $element (@x) {
    print "$element\n";
}
```

# Loops: foreach

- foreach lets you modify the original array

```
foreach my $element (@x) {
    # This actually changes @x!
    $element = 'Hello';
}
```

# Loops: each

- Foreach works on a hash

```
foreach my $key (keys(%x)) {
  print "$key maps to $x{$key}\n";
}
```

- But sometimes it's easier to say

```
while(my($key, $val) = each(%hash))
{
  print "$key maps to $val\n";
}
```

# Other Languages

# Perl

- Arrays

```
@futurama = ( "Bender", "Fry" );
$futurama[1]
```

- Hashes

```
%series = (
    "Dark Tower" => "King",
    "Harry Potter" => "Rowling");
$series{"Harry Potter"}
```

# Ruby

- Arrays

```
futurama = [ "Bender", "Fry" ]
futurama[1]
```

- Hashes

```
series = {
   "Dark Tower" => "King",
   "Harry Potter" => "Rowling"}
series["Harry Potter"]
```

# Python

- Arrays (lists)

```
futurama = [ "Bender", "Fry" ]
futurama[1]
```

- Hashes (dictionaries)

```
series = {
   "Dark Tower" :  "King",
   "Harry Potter" :  "Rowling"}
series["Harry Potter"]
```

# Compared: Array Size

- Perl: `scalar(@array)`
- Python: `len(array)`
- Ruby: `array.length`
- Javascript: `array.length`

# Compared: Remove and return last item

- Perl: `pop(@array)`
- Python: `array.pop`
- Ruby: `array.pop`
- Javascript: `array.pop()`

# Merging arrays and hashes

- Lua, JavaScript, PHP, and others have one type for both
  - Lua: tables
    - `a = {}`
    - `a["bob"] = "barker"`
    - `a[1] = "steak sauce"`
  - Javascript: array
    - `var a = new Array();`
    - `a["bob"] = "barker";`
    - `a[1] = "steak sauce";`

# Look for variations

- Python offers a native "set"

```
spammers = set([ \
  'malware@example.com',
  'scammer@example.org'])
if email in spammers:
    print "Refusing SPAM\n"
```

# Look for variations

- PHP preserves insert order!

```
$arr[2] = "two";
$arr[3] = "three";
$arr[1] = "one";
foreach ($arr as $element) {
  print "$element ";
}
```

- prints: "two three one "

# Homework

- Implementing Metacritic or Rotten Tomatoes
  - Collect reviewers scores
  - Report the scores and an average