Perl Data Structures

Follow along with the PDF!
http://perlcabal.org/~util/YAPC/YAPC_2011_data_structures/

YAPC::NA   2011-06-27

http://www.yapc2011.us/yn2011/talk/3235

Util <bruce.gray@acm.org>

For posterity

Since we want to cover...

Stacks, Queues, Hogwarts, Dictionaries, Records, Rocky Horror, Sets, Bags, Cheech and Chong, Power Tools, Slicing off your fingers, DBI, The Princess Bride, SQLite, AoA, AoH, HoA, HoH, Mountain Climbing, Auto-vivification*

* Apologies to TheDamien

No...

- doopri...
-

20 Minutes!

Bonus Slides

Assumes you know "baby Perl"

- Basic control flow
  - if (...) {...} elsif (...) {...} else {...}
  - while (...) {...}
  - for (...) {...}
- Basic variable use
  - $num = 3;   say $num;
  - @nums = (3, 4, 5, 6);   say $nums[0];
  - %bunch = ('Brady' => 6);   say $bunch{'Brady'};

I use say

- Perl 5.10 was released 5 *years* ago.
- use 5.010;   say 'Profit!';
- perlbrew can install your own custom build of recent Perls.
- or, change: `say    $my_name;`
  to: `print $my_name, "\n";`
  and remove: `use 5.010;`

Atlanta.pm says:

- Hashes are wonderful
- All the languages that have hashes are more powerful by far than all the languages that don't.
- And the ones that don't, that are in current growth modes, are adding hashes.

Well Said

- "Show me your code and conceal your data structures, and I shall continue to be mystified. Show me your data structures, and I won't usually need your code; it'll be obvious." [1]
  -- Frederick P. Brooks, "The Mythical Man-Month"
- I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.
  -- Linus Torvalds, the Highly Opinionated [2]

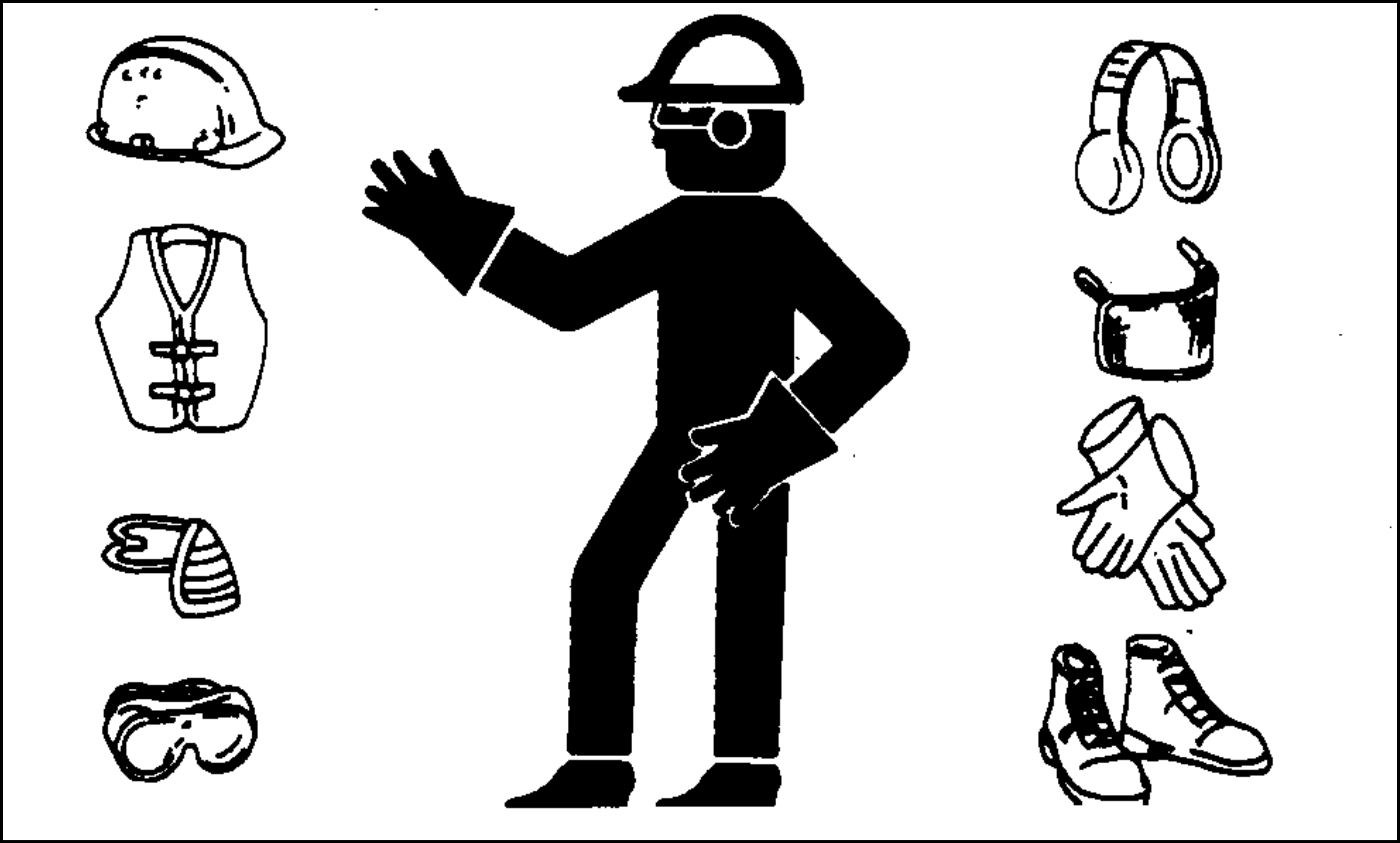
1 Actually, Brooks said "Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious."
2 For a fun time, watch Linus speaking about Git, and ranting about Subversion being the dumbest project in the history of software. http://www.youtube.com/watch?v=4XpnKHJAok8

Power Tools

```
#!/usr/bin/perl
use strict;
use warnings;
#use Data::Dumper; $Data::Dumper::Useqq = 1; $Data::Dumper::Sortkeys = 1; $| = 1;

=begin comment

2011-xx-xx  Bruce Gray
Wrote program.

This program ...XXX

=end comment

=cut

my $file = '';
push @ARGV, $file unless @ARGV;

while (<>) {


}



__END__
```

```perl
use strict;
use warnings;
#use Data::Dumper; $Data::Dumper::Useqq = 1; $Data::Dumper::Sortkeys = 1; $| = 1;
```

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



```
1.  use Data::Dumper;

2.  $Data::Dumper::Useqq    = 1;
3.  $Data::Dumper::Sortkeys = 1;
4.  $|                      = 1;
```

1. Loads the Data::Dumper module.
2. Tells Dumper to show you the unprintables as \t, \n, etc.
3. Tells Dumper to sort hashes by key asciibetically.
4. Turns off buffering, so you see output in real-time.

Slows your program, so keep **commented out** until needed.

Nouns vs Verbs
Singular vs Plural Nouns

Plural variable types

	Ordered	Key	Duplicate Keys Allowed?	Sigil	Values are
Array	Yes	Position [Integer]	No	@	Scalars
Hash	No	Name {'String'}	No	%	Scalars

Array indexes start at 0.
Scalars can be numbers, strings, undef, references, objects, etc.

Atlanta.pm says:

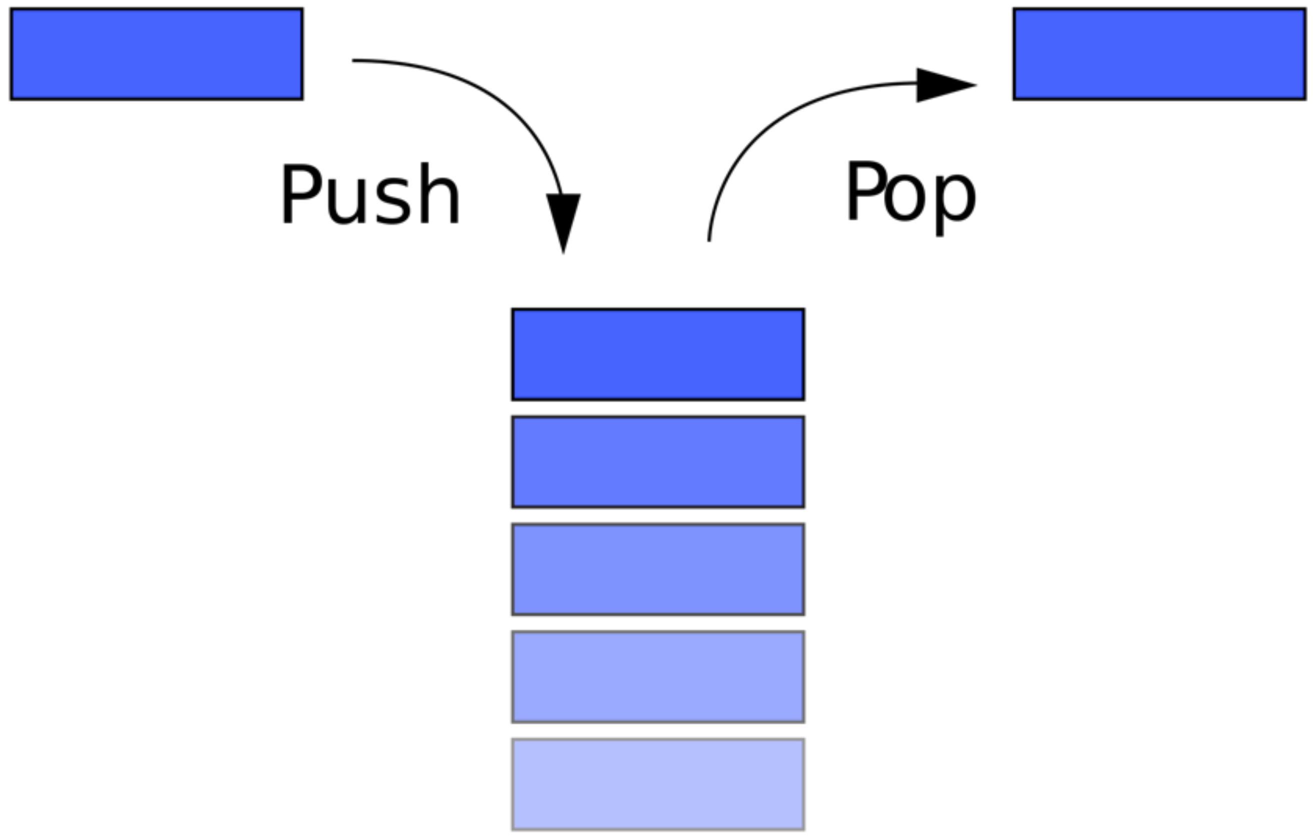
- Use physical props!

Arrays&Hashes + Functions = Profit!

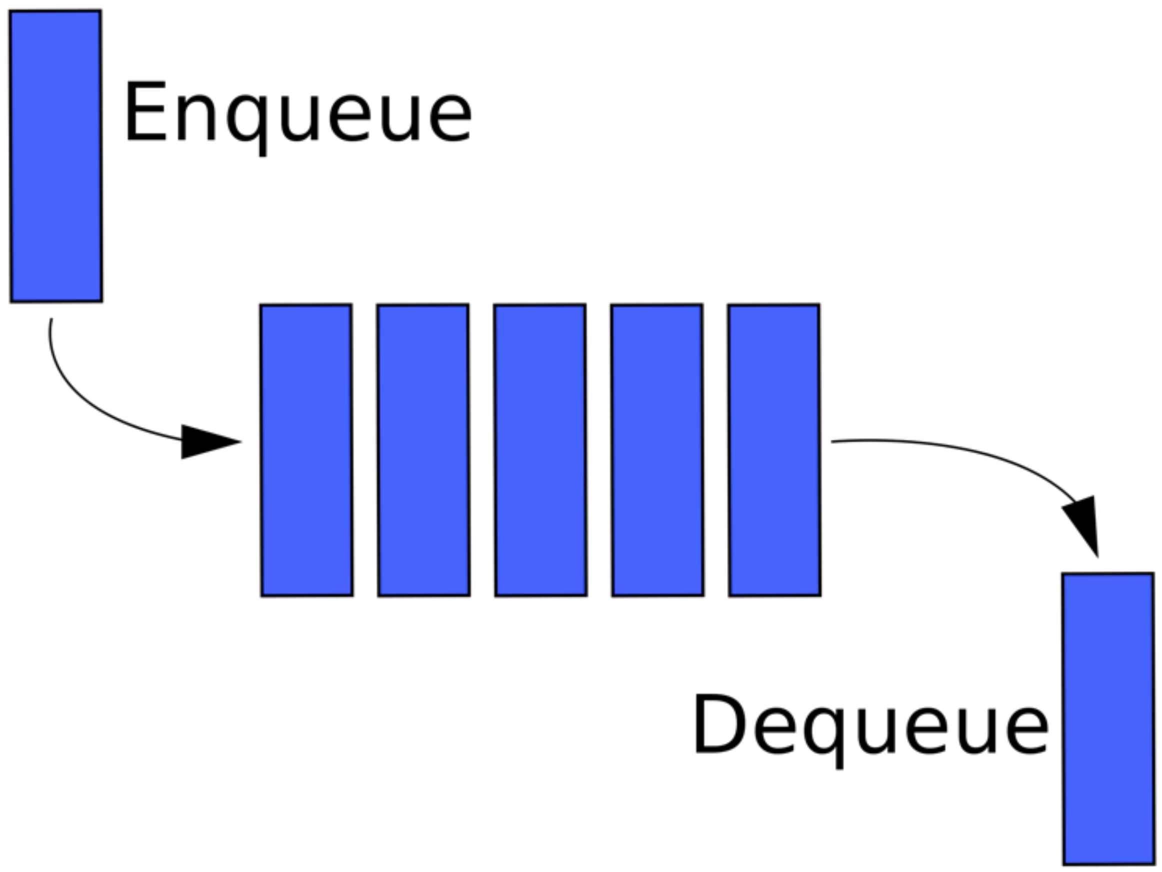
Functions for Arrays

```
              Head[0] ............. Tail[-1]

  Unshift -->                                      <-- Push

    Shift <--                                      --> Pop

              ^^^^^^^^^^^ Splice ^^^^^^^^^^^^^
```

- Unshift & Push ***receive*** an element, and put it into the array, making it larger.
- Pop & Shift take an element out of the array, shrinking it, and ***return*** the element.



Stack (LIFO)



Queue (FIFO)

"You keep using that word;
I don't think it means what
you think it means"

Stacks & Queues

	Add with	Remove with	Order
Stack	Push	Pop	LIFO
Queue	Push	Shift	FIFO

Data Structure: Set

Janet & Rocky are surprised

by Dr. Scott, Frankie, and Brad

Set

```
chomp( my @lines = <DATA> );
my %seen;
for my $thing ( @lines, @lines, @lines ) {

    next if $seen{$thing};
    $seen{$thing} = 1;

    print $thing, ' ';
}

# Output: Janet! Dr. Scott! Brad! Rocky! Ugh!

__DATA__
Janet!
Dr. Scott!
Janet!
Brad!
Rocky!
Ugh!
```

Atlanta.pm says:

- Data::Dumper is the greatest thing under the sun if you come from a language that does not have it.
- When something seems wrong with your data, you can just **look** at it!
- D::D is so powerful that many professional Perl programmers don't know how to use the debugger, because they never***need** it.

Data Structure: Bag

```
use strict;
use warnings;
use Data::Dumper; $Data::Dumper::Useqq=1; $Data::Dumper::Sortkeys=1;

push @ARGV, '/usr/share/dict/words' unless @ARGV;



my %h;
while (<>) {
    my $first_letter = lc substr $_, 0, 1;
    $h{$first_letter}++;
}



print Dumper \%h;
```

```
$VAR1 = {                             "n" => 6748,
    "a" => 17061,                     "o" => 7835,
    "b" => 11027,                     "p" => 24412,
    "c" => 19861,                     "q" => 1152,
    "d" => 10843,                     "r" => 9596,
    "e" => 8716,                      "s" => 25031,
    "f" => 6854,                      "t" => 12909,
    "g" => 6836,                      "u" => 16386,
    "h" => 8984,                      "v" => 3411,
    "i" => 8787,                      "w" => 3927,
    "j" => 1571,                      "x" => 385,
    "k" => 2220,                      "y" => 671,
    "l" => 6233,                      "z" => 949
    "m" => 12531,    };
```

Data Structure: Dictionary

```
use strict;
use warrnings;
use 5.010;
my %founder_of_house;
while (<DATA>) {
    chomp;
    my ( $house, $founder ) = split;
    $founder_of_house{$house} = $founder;
}

say $founder_of_house{'Ravenclaw'};           # Output: Rowena

__DATA__
Gryffindor   Godric
Hufflepuff   Helga
Ravenclaw    Rowena
Slytherin    Salazar
```

Note: one value only; assigning again replaces the value

```
...

$founder_of_house{'Ravenclaw'} = 'Helena';
say $founder_of_house{'Ravenclaw'};

# Output: Helena
```

Record

```
# Teaser code - does not work

while ( my %person = read_database_record() ) {
    say $person{'NAME'}, ' lives at ',
        $person{'ADDRESS'};
}
```

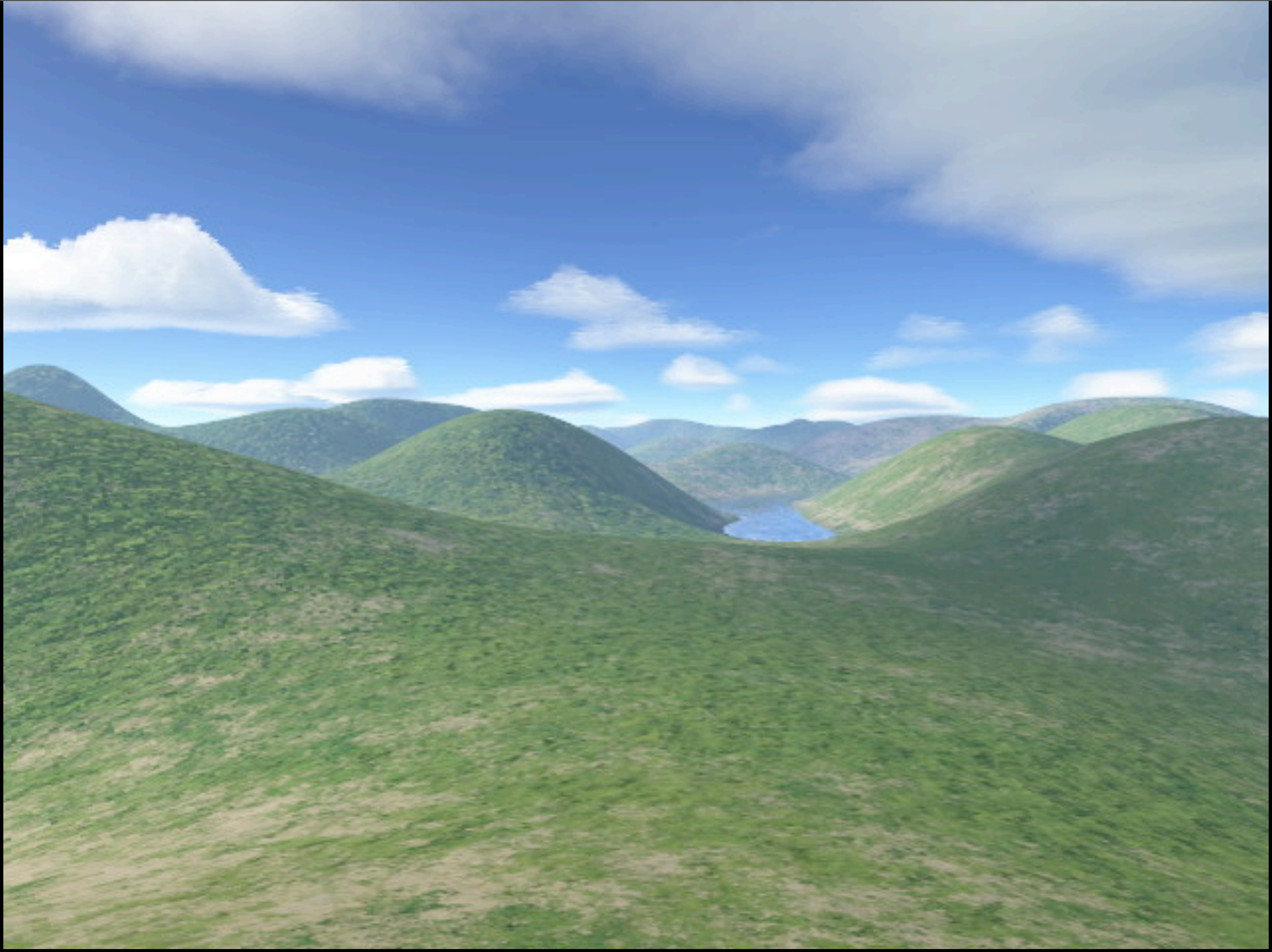
Records make more sense when there are more of them, so more on that when we do Multi-level.

Records vs Dictionaries

- Dictionaries have keys that vary, and values that vary.
- Records have values that vary, but use fixed keys, usually hard-coded.

Clear so far?

The most important idea in this talk...



The other side of the hill

- We don't have a good way to tell people that.

- Multi-level Data Structures
- Regular Expressions

Multi-level Data Structures

Just a little new syntax

`\`   `->`   `sigil{}`

Use that syntax

```
\@array          # Get a reference to an array
\%hash           # Get a reference to a hash

$aref->[3]       # Dereference one array element
$href->{'NAME'} # Dereference one hash value

@{$aref}         # Dereference all. @$aref works too.
%{$href}         # Dereference all. %$href works too.
```

But not **that** syntax

```
$$arrayref[3]

$$hashref{'SomeKey'}
```

Know it, in case you see it in other people's code, but please don't use it in your own.
See Perl Best Practices 11.2 - Braced References

This is why you need `use strict`

If you type `$href{'SomeKey'}` when you should have typed `$href->{'SomeKey'}`, you are referring to a whole hash named `%href` .

Without `use strict`, Perl will create `%href` for you! Ack!

Database setup

```
rm      yapc_ds_talk.db
sqlite3 yapc_ds_talk.db
CREATE TABLE registrations ( username CHAR(10), regtype
CHAR(2), active CHAR(3) );
INSERT INTO  registrations VALUES( 'Able'   , 'ch', 'no'  );
INSERT INTO  registrations VALUES( 'Baker'  , 'ch', 'yes' );
INSERT INTO  registrations VALUES( 'Charlie', 'ch', 'no'  );
INSERT INTO  registrations VALUES( 'Roger'  , 'ch', 'no'  );
INSERT INTO  registrations VALUES( 'Fox'    , 'zz', 'yes' );
INSERT INTO  registrations VALUES( 'Dog'    , 'ch', 'no'  );
.quit
```

SQLite Rocks!

http://www.sqlite.org/

```
#!/usr/bin/perl
use strict;
use warnings;
use 5.010;
use Data::Dumper; $Data::Dumper::Useqq = 1; $Data::Dumper::Sortkeys = 1;
use DBI;


my $sqlite_db_name = 'yapc_ds_talk.db';
my $dsn = "dbi:SQLite(RaiseError=>1):dbname=$sqlite_db_name";
my $dbh = DBI->connect($dsn) or die;

my $aref = $dbh->selectall_arrayref(
    'SELECT * from registrations',
    { Slice => {} },
);

print ' ', $_->{username} for @{$aref};
print "\n";

for my $href ( @{$aref} ) {
    print Dumper $href;
}
$dbh->disconnect;
```

```
Output:
Able Baker Charlie Roger Fox Dog

$VAR1 = { "active" => "no",   "regtype" => "ch",  "username" => "Able"    };
$VAR1 = { "active" => "yes",  "regtype" => "ch",  "username" => "Baker"   };
$VAR1 = { "active" => "no",   "regtype" => "ch",  "username" => "Charlie" };
$VAR1 = { "active" => "no",   "regtype" => "ch",  "username" => "Roger"   };
$VAR1 = { "active" => "yes",  "regtype" => "zz",  "username" => "Fox"     };
$VAR1 = { "active" => "no",   "regtype" => "ch",  "username" => "Dog"     };
```

AoWhat?

• AoA	Array of Arrays (or List of Lists)
• AoH	Array of Hashes
• HoA	Hash of Arrays
• HoH	Hash of Hashes
• AoHoHoA	Array of Hashes of Hashes of Arrays

Atlanta.pm says:

- Who could conceptualize more than 3 levels?

AoHoHoA

- A Hospital
    - has multiple numbered floors
        - with multiple wards/units per floor
            - with multiple nurses per ward
                - with a list of patients, in visitation order

```
$hospital->[3]{'MedSurg'}{'Sarah'}[4];
```

AoHoHoA

```
for my $floor ( sort { $a <=> $b } keys
$hospital->[3]{'MedSurg'}{'Sarah'}[4];
```

Auto-vivification

http://perlmonks.org/?node_id=172833

The Two Auto-Vivifies (in reply to "re-using a hash reference")

"Keep in mind that even ***checking for the presence of*** *a hash key that does not exist will auto-vivify it into existence. "*

Almost :)

Perl will auto-vivify the hashes/hashkeys *necessary* to evaluate an expression.

This will clarify the differences:

```
perl -MData::Dumper -we 'print Dumper $h1 unless $h1->{a}->{b}->{c}'
$VAR1 = {
          'a' => {
                   'b' => {}
                 }
        };
```

In order to test for $h1->{a}->{b}->{c} there must exist a hash for 'c' to be a key in, so Perl auto-vivifies the data structure up to that point. But notice that 'b' points to an empty hash; 'c' is not a key in it! Perl sees that 'c' is not in {'b'}, and that is all that it needs to evaluate the unless.

```
perl -MData::Dumper -we 'print Dumper $h1 for $h1->{a}->{b}->{c}'
$VAR1 = {
          'a' => {
                   'b' => {
                            'c' => undef
                          }
                 }
        };
```

Here, we asked Perl to actually *obtain* the value stored in 'c', so Perl went one step further to create the hash key 'c', which holds 'undef', which it returns to the for statement.

Auto-vivification

- will bite you sometimes, but it much better than having to do without it. Trust me, I have worked in languages that make you do it all by hand.

Resources

http://perldoc.perl.org/perldata.html	Perl data types
http://perldoc.perl.org/perlref.html	Perl references and nested data structures
http://perldoc.perl.org/perlreftut.html	Short tutorial about references
http://perldoc.perl.org/perllol.html	Manipulating Arrays of Arrays in Perl
http://perldoc.perl.org/perldsc.html	Perl Data Structures Cookbook

Thanks to Atlanta PerlMongers, who previewed this talk. Their feedback greatly enhanced the final form.

Have fun with your new toys!