Around the PERL in 11 days

Two Dimensional Arrays
References

• A reference can be thought of as the directions for finding a variable
• Use \ to get a reference to a variable
• $ref = \$name
  - $ref is a scalar variable that holds a reference to the scalar $name
• $ref = \@names
  - $ref is a scalar variable that holds a reference to the array @names
2D Matrices

- If @matrix is a list of references to other lists, then @matrix is a 2D matrix.
- De-reference a reference by wrapping it in {} and the $, @, or % depending on what it is.
- $matrix[0]$ is a reference to the first list.
- @{$matrix[0]} is the first list.
  \- @new_list = @{$matrix[0]};
- @{$matrix[0]}[0] is the first thing in the first list.
  \- $bob = @{$matrix[0]}[0];
Matrix Shorthand

- Rather than use the messy \{$\{ \ldots \} \}$[1] wrapping around a reference to a list you can just put another index on the end
  - @matrix is a list of references to lists
  - \{\{$matrix[0]\}\}[0] and $matrix[0][0]$ are the same
2D Matrices in Subroutines

• If you are working with 2D matrices in a subroutine, you might have been passed a reference to @matrix

• Now you need to dereference two references:

```perl
$ref = \@matrix
my_function($ref);
sub my_function {
    $upper_left_cell = $$ref[0][0]
}
```

```perl
$$ref[0][0] is shorthand for

```perl
${${$ref}[0]}[0] (a horrible mess)
for($j=0; scalar(@matrix);$j++) {
    print "$matrix[$j][0]";
    for($i=1; scalar(@{$matrix[$j]});$i++) {
        $matrix[$j][$i] = 5;  #Make all cells 5
    }
    print "\n";
}
Printing 2D Arrays

```perl
for ($j = 0; scalar (@matrix); $j++) {
    print "\$matrix[\$j][0]";
    for ($i = 1; scalar (@{\$matrix[\$j]}); $i++) {
        print "\t\$matrix[\$j][\$i]";
    }
    print "\n";
}
```
Summary

• if @matrix is a list of references to other lists:
  
• $\text{matrix}[0][1]$ is the second thing in the first row

• @{$\text{matrix}[0]}$ is the whole first row