use strict AND my

- With “use strict”, variables must be declared with “my”
- More work at first, but saves pain later!
  - Avoids errors from same vars being used in diff. files

```perl
use strict;
use CGI qw(:standard);
print header();

my($w)  = (87);
my($x)  = 89;
my($y, $z) = (91, 93);
my(@array) = (1, 2, 3);
my($d1, $d2, $d3) = @array;
my($f1, @f2, $f3) = @array;

print p("w is $w");
print p("x is $x");
print p("y is $y");
print p("z is $z");
print p("d1: $d1 d2: $d2 d3: $d3");
print p("f1: $f1 f2: @f2 f3: $f3");

my($details) = "John|rabbit7";
my($name,$password) = split (/\|/,$details);
print p("name: '$name' password: '$password'");
print(end_html());
```
Perl Function Calls ("subroutines")

```perl
use CGI qw( :standard );
use strict;
print( header() );

# Prints "hello", takes no arguments
sub hello {  
    print "\n<br/> Hello.";
}

# Takes two arguments, return their product
sub multiply {  
    my($valA, $valB) = @_;  
    return $valA * $valB;
}

my($x) = 2;
&hello;
print "\n<br/> $x * 7 = " . &multiply($x,7);
&hello();
&hello(72145);
print(end_html());
```

Function Calls and Arrays

```perl
# Takes an array as argument, returns minimum value
sub findMin {  
    my(@array) = @_;  
    my $min = @array[0];  
    my $ii;  
    my $len = @array;
    for ($ii=0; $ii < $len; $ii++) {  
        if (@array[$ii] < $min) {  
            $min = @array[$ii];  
        }
    }
    return $min;
}

# Defines new global array, @array1
# AND returns a new array with 4 elements.
my @array1 = ();
sub makeArray() {  
    @array1 = (89, 23, 90);  
    my @array2 = (34, 5.4, 123, 2.01);  
    return @array2;
}

@test1 = makeArray();
@test2 = (89, 23, 40, -17);
print "\nMin1 is: " . &findMin(@test1);
print "\nMin2 is: " . &findMin(@test2);
print "\nMin3 is: " . &findMin(@array1);
print "\nMin4 is: " . &findMin(@array2);
```
Exercise #1

- Write a Perl function checkNum() that takes three arguments, num, min, and max, and returns 1 if num is in the range [min, max] (inclusive), or 0 otherwise.

Exercise #2

- Write a function dup() that takes two arguments, ch and count, and prints the value of ‘ch’ out count times.
- Then write code to produce the following output:
  12 12 12 12 12
Exercise #3

- Write a function, makeArray, that takes one argument, count, and returns an array of size count with the numbers from [1..count]. So makeArray(4) should return (1, 2, 3, 4)

Exercise #4

- Write a Perl function, reverse(), that takes one argument, an array, and returns that array in reverse order. So (1, 2, 3) becomes (3, 2, 1).
String → number conversions (and back)

- Perl will convert to number where needed, or to a string where needed

```perl
my $str1 = "27";
my $str2 = "dog";
my $str3 = "cat";

my $result1 = $str1 + 10;
my $result2 = $str1 - 10;
my $result3 = $str2 + 10;

print p("result1: $result1 result2: $result2");
print p("result3: $result3");

my $val1  = 13;
my $val2  = 27;

print p("Combine these: " . $val1 . $val2);

if ($str2 == $str3) {
  print h2("Dogs and cats unite!");
}
```

Gotchas, References, and Multiple Files

```perl
my(@array) = @_;  # not the same as
my(@array) = $_;

my ($valA, $valB) = @_;  # not the same as
my $valA, $valB = @_;

References:
@array = (1, 2, 3);
$ref_array = \@array;
$array2 = @$ref_array;

print "\nfrom ref: " . $$ref_array[1];
print "\nfrom array2: " . $array2[1];

Multiple Perl Files:
require "question_struct.pl";
Be sure not to use same names (e.g., function names) in different files!
```
elsif

if ($x > 0) {
    print "Hello";
}
elsif ($x == -5) {
    print "Goodbye";
}
else {
    print "Bye";
}