

# IT350 Web and Internet Programming

## SlideSet #13: Perl Functions and More

(see online references)

### use strict AND my

- With “use strict”, variables must be declared with “my”
- More work at first, but saves pain later!

```
#!/usr/bin/perl
use strict; use CGI qw( :standard );
use CGI::Carp qw(warningsToBrowser fatalitiesToBrowser);
print header(); print start_html();

my $x      =  89;
my ($y, $z) =  (91, 93);

my @arr =  (1, 2, 3);

my($d1, $d2, $d3) = @arr;
my($f1, @f2, $f3) = @arr;

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”)

```
#!/usr/bin/perl
use strict;
use CGI qw( :standard );
use CGI::Carp qw(warningsToBrowser fatsalsToBrowser);

print header(); print start_html(); print ('<p>');

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

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

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

## Function Calls and Arrays

```
... #usual prelude here

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

my @test1 = makeArray();
my @test2 = (89, 23, 40, -17);
print "\n<br />Min1 is: " . findMin(@test1);
print "\n<br />Min2 is: " . findMin(@test2);
print "\n<br />Min3 is: " . findMin(@array1);
print "\n<br />Min4 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

```
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

```
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 array: " . $array[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";
}

```

## Regular Expressions and Matching Operator

---

```

1 #!C:\Perl\bin\perl
2 # Fig 25.7: fig25_07.pl
3 # Searches using the matching operator and regular expressions.
4
5 $search = "Now is is the time";
6 print( "Test string is: '$search'\n\n" );
7
8 if ( $search =~ /Now/ ) {
9     print( "String 'Now' was found.\n" );
10 }
11
12 if ( $search =~ /^Now/ ) {
13     print( "String 'Now' was found at the beginning of the line." );
14     print( "\n" );
15 }
16
17 if ( $search =~ /Now$/ ) {
18     print( "String 'Now' was found at the end of the line.\n" );
19 }
20
21 if ( $search =~ /\b (\w+ \w+) \b/x ) {
22     print( "Word Found ending in 'ow': $1\n" );
23 }
24
25 if ( $search =~ /\b (\w+ )\w (\w+) \b/x ) {
26     print( "Repeated words found: $1 $2\n" );
27 }
28

```

---

## Regular Expression Quantifiers and Metacharacters

Quantifier	Matches
{n}	Exactly n times
{n, n}	Between n and n times inclusive
{n, }	n or more times
*	One or more times (same as {1, })
*	Zero or more times (same as {0, })
?	One or zero times (same as {0,1})

Fig. 25.8 Some of Perl's quantifiers.

Symbol	Matches	Symbol	Matches
^	Beginning of line	\d	Digit (i.e., 0 to 9)
\$	End of line	\D	Nondigit
\b	Word boundary	\s	Whitespace
\B	Nonword boundary	\S	Nonwhitespace
\w	Word (alphanumeric) character	\n	Newline
\W	Nonword character	\t	Tab

Fig. 25.9 Some of Perl's metacharacters.