ECOL 553L

Perl Subroutines

adapted from: http://oreilly.com/catalog/lperl3/chapter/ch04.html
Subroutines

• Pieces of script that are “inserted” at a location when its called
• have their own scope to define variables if needed
  • can access global variables
• written to do things that we need to do often (and maybe change a little)
Defining a subroutine

• The keyword `sub` starts a routine definition
• it is IMMEDIATELY followed by the subroutine name

```perl
my $n = 0;

sub student {
    $n += 1;  # Global variable $n
    print "Hello, student number $n!\n";
}
```
Invoking a subroutine

• Also called calling
• two methods (if no operators)
  • <name>;
  • <name>();
• both the same

my $n = 0;

sub student {
    $n += 1;  # Global variable $n
    print "Hello, student number $n!\n";
}
student;
student;
student;
student;
Returning a value

- Sometimes we want subroutine to pass back a value from its invocation
- We do this with the `return` keyword, followed by a variable or value
  - `return $n;`
  - `return 4+7;`

```
my $n = 0;

sub student {
    $n += 1;  # Global variable $n
    print "Hello, student number $n!\n";
    return $n;
}

my %nameID;
foreach my $name (@ARGV){
    $nameID{$_} = student;
}
```
Passing arguments

• sometimes you want to take an argument from the call
• these values come into the subroutine using the @_ special variable
• you pass them just like any other function
  • <name>(<arg1>,<arg2>,....);

```perl
sub max {
    if ($_[0] > $_[1]) {
        return $_[0];
    } else {
        return $_[1];
    }
}
max(1,2);
max(4,-9);
max(4e3,4001);
```
Private variables

- You can define variables that only exist in the single instantiation of the subroutine

```perl
sub max {
    my($a, $b);       # new, private variables for this block
    ($a, $b) = @_;    # give names to the parameters
    if ($a > $b) { $a } else { $b }
}
```
What size is @_?

• Turns out @_ can be larger or smaller than expected
• In our previous version of max, the call to max(1,2,3) would not throw an error, but would return a wrong answer
• How would we fix that?
What size is @_?

- Turns out @_ can be larger or smaller than expected
- In our previous version of max, the call to max(1,2,3) would not throw an error, but would return a wrong answer
- How would we fix that?

Solution 1: Error

```perl
sub max {
    if (@_ != 2) {
        die "WARNING! max should get exactly two arguments!\n";
    }
    if($_[0]>$_[1]) { return $_[0]; } else { return $_[1]; }
}
```
What size is @_?

• Turns out @_ can be larger or smaller than expected
• in our previous version of max, the call to max(1,2,3) would not throw an error, but would return a wrong answer
• How would we fix that?

**Solution 2: Iterative**

```perl
sub max {
    my($max_so_far) = shift @_;  # the first one is the largest yet seen
    foreach (@_) {               # look at the remaining arguments
        if ($_ > $max_so_far) {   # could this one be bigger yet?
            $max_so_far = $_;
        }
    }
    return $max_so_far;
}
```
What size is `@_`?

- Turns out `@_` can be larger or smaller than expected
- In our previous version of max, the call to `max(1,2,3)` would not throw an error, but would return a wrong answer
- How would we fix that?

**Solution 3: Recursive**

```perl
sub max {
    my $a = shift @_;         
    my $b = shift @_;         
    my $max = $a;             
    if($b > $a){ $max = $b; } 
    if(scalar(@_) > 0){       
        unshift @_, $max;     
        $max = max(@_);      
    }
    return $max;
}
```