Programming in Perl
Lecture 6 Examples

Creation of Hard References

```
-- backslash operator (functions much like & in C/C++)

$ref_scalar = \$scalar;
$ref_array = \@array;
$ref_hash = \%hash;
$ref_sub = \&subname;

-- ANONYMOUS ARRAY COMPOSER

$ref_array = [$a, $b, $c];
# referenence to an unnamed array consisting of
# of the values of $a, $b, and $c at the time of creation

# contrast with
@array_of_refs = \($a, $b, $c);
# a three element array containing references
# to $a, $b, and $c
# same as
@array_of_refs = (\$a, \$b, $\c);

-- ANONYMOUS HASH COMPOSER

$ref_hash = { # note the brackets!
    lunch => 'noon',
    dinner => 'six pm'
};
```

Using Hard References

```
$c = 3;
$ref_scalar = \$c;
# rightmost $ dereferences, leftmost $ indicates scalar value
$value = $$ref_scalar; # $value is 3
# could also write
$value = ${$ref_scalar}; # same as above
@ref_array = [1, 2, 3];
$$ref_array[2] = 4;
# the following produces the same result
# it says $ref_array is a reference;
# dereference it and return the second element
# of the result (it always defaults to a scalar
# context -- i.e., always returns a scalar value)
$ref_array->[2] = 4;
print("@$ref_array\n"); # prints 1 2 4<newline>
$$ref_hash{'breakfast'} = 'seven am';
@meals = keys(%$ref_hash);
$c = &$ref_sub(2);  # calls a sub
```

To illustrate the difference between [and \(:

```
$a = 1; $b = 2; $c = 3;

$ref_array = [$a, $b, $c];
# assign 100 to the first array location
$$ref_array[0] = 100;
print("@$ref_array\n");
print("$a\n");
# 100 2 3
# 1

@array_of_refs = \($a, $b, $c);
# note $$array_of_refs[0] == ${$array_of_refs}[0],
#   which would be incorrect because @array_of_refs
#   is NOT a reference; therefore, must use {}
${$array_of_refs[0]} = 100;
# assign 100 to the location referred to
# by the first element of the array
foreach $ref (@array_of_refs) {
    print ($$ref, " ");
}
print("\n");
print("$a\n");
# 100 2 3
# 100
```

Passing References as Parameters

```
      ($ref_first, $ref_second) = order_by_means(\@a, \@b);
      print("First: @$ref_first\n");
      print("Second: @$ref_second\n");

      sub order_by_means {
         my($ref_a) = $_[0];
         my($ref_b) = $_[1];

         my($mean_a) = calc_mean($ref_a);
         my($mean_b) = calc_mean($ref_b);

         if ($mean_a > $mean_b) {
           return ($ref_b, $ref_a);
         }
         else {
           return ($ref_a, $ref_b);
         }
     }
```

```
sub calc_means {
    my($ref_array) = $_[0];

    my($sum) = 0;

    if ( @$ref_array == 0 ) {
return 0;
    }
    foreach $elem (@$ref_array) {
      $sum += $elem;
    }
    return ( $sum / @$ref_array ) ;
}
```

Passing Filehandles as Parameters:

```
o  special syntax used (typeglob)

$rec = get_rec(\*STDIN);
sub get_rec {
  my($handle) = $_[0];
  return (scalar <$handle>);
}
```

BACK TO CGI...

The Class CGI

```
# Adapted from Lincoln D. Stein.,
# How To Set Up and Maintain a Web Site

use CGI qw(:standard);

$query = CGI->new();
print $query->header;
print $query->start_html('Skeleton Script');

# note this isn't a function call,
# this is a hash which is a data
# item in the object $query
unless ($query->param) {
   &display_prompt($query);
} else {
   &process_query($query);
}

&display_signature;
print $query->end_html;
```

```
# --------------------- subroutines --------------------
# print the prompt
sub display_prompt {
    my ($query) = $_[0];
    print $query->h1("Skeleton Script");
    print $query->hr();
    print $query->start_form;
    print $query->p("Type something: ",$query->textfield('something'));
    print $query->submit;
    print $query->end_form;
}

# do the processing
sub process_query {
    my($query)=$_[0];

    my($value) = $query->param('something');
    print "The value returned is $value\n";
    print p("<STRONG>Your code here!</STRONG>"), "\n";
}

# print a canned signature.
sub display_signature {
print <<END;
<HR>
<ADDRESS>Dead letters office</ADDRESS>
<P>Last modified: April 28, 1996
<P><A HREF="/">Home Page</A>
END
```

More Verbose Calling Sequences

```
      -- instead of

    print $query->p("Type something: ",$query->textfield('something'));

           we could have written

    print $query->p("Type something: ",
                    $query->textfield( -NAME => 'something' ));

      -- the -NAME is called a named parameter; what follows the => is
called a parameter value

      -- these types of actual argument lists allow arguments to be
passed in any order while being more descriptive

      -- of course you have to know the proper literal names to use
named parameters
```

Example of Named Calling Sequence

```
print $query->scrolling_list(
        -NAME => "flavors",
        -VALUES => [ qw(mint chocolate cherry vanilla peach) ],
        -LABELS => {
           mint => "Mighty Mint",
           chocolate => "Cherished Chocolate",
           cherry => "Cheery Chery",
           vanilla => "Very Vanilla",
           peach => "Perfectly Peachy",
         },
        -SIZE => 3,
        -MULTIPLE => 1,
      );

     -NAME : string which can be used to retrieve user data from the
form with param()

     -VALUES : a reference to an array which contains a list of the
possible values passed back to the CGI script (accessible using
param() with the appropriate name string)

     -LABELS : a reference to a hash which relates the values defined
above to the labels actually displayed to the user in the scrolling
list (optional)

     -SIZE : an integer giving the number of list items displayed to
the user at one time (optional)

     -MULTIPLE : set to 1 (true) to allow the user to choose more than
one item (false means the user can select only one item)

     -DEFAULT : [not used here] selects values that are highlighted as
already selected when the user first looks at the list (optional)
```

Instead on anonymous hash...

```
%flavors = (
           mint => "Mighty Mint",
           chocolate => "Cherished Chocolate",
           cherry => "Cheery Chery",
           vanilla => "Very Vanilla",
           peach => "Perfectly Peachy",
);

print $query->scrolling_list(
        -NAME => "flavors",
        -LABELS => \%flavors,
        -VALUES => [ keys %flavors ],
        -SIZE => 3,
        -MULTIPLE => 1,
      );
```

Example List of Widgets and Parameters

```
  -- TEXTFIELD

    o prompts for a single line of text input

print $query->p("Name:",$query->textfield(
                           -NAME => 'something',
                           -DEFAULT => 'unknown',
                           -SIZE => 20,
                           -MAXLENGTH => 80
                        )
               );

-NAME : as for scrolling list
-DEFAULT : starting value displayed in field (optional)
-SIZE : size of the field in characters (optional)
-MAXLENGTH : maximum number of characters accepted (optional)


  -- TEXTAREA

    o prompts for text input, allowing multiple lines

    o displays a rectangular box to enter text into

print $query->p("Anything to say for yourself?", br(),
                $query->textarea( -NAME => 'comments',
                                  -DEFAULT => "A while back...",
                                  -ROWS => 10,
                                  -COLUMNS => 50
                                )
                ), "\n";

-NAME : as for scrolling list
-DEFAULT : as for textfield (optional)
-ROWS : height in number of lines the text area  will occupy
-COLUMNS : width in number of characters  the text area  will occupy
```

```
  -- STANDALONE CHECKBOXES

    o creates an isolated checkbox

    o user can check the box or leave it unchecked

print $query->p($query->checkbox( -NAME => 'sample',
  -CHECKED => 1,
  -VALUE => 'YES',
  -LABEL => "Free sample"
)
), "\n";

-NAME : as for scrolling list
-CHECKED : 1 if the box should start as checked (optional)
-VALUE : value of the checkbox when checked; default is 'on'
(optional)
-LABEL : label to be attached to the box for the user to see ;
default is the name of the box (optional)

  -- CHECKBOX GROUP

    o creates a list of logically related checkboxes

    o any number can be checked or unchecked at once

print $query->p($query->checkbox_group(
                  -NAME => 'operating_systems',
                  -VALUES => [ 'IRIX', 'AIX', 'Solaris', 'Sun OS' ],
                  -DEFAULT => [ 'AIX' ],
  -LINEBREAK => 1,
 )
   ), "\n";


-NAME : as for scrolling list
-VALUES : as for scrolling list
-DEFAULT : reference to a list containing values to be checked at
start or a single value to be checked; default is that nothing is
selected initially (optional)
-LINEBREAK : set to 1 if boxes are to be displayed on separate
lines; default is single line (optional)
-LABELS : [not used above] as for scrolling list (optional)
```

```
    -- RADIO GROUP

      o creates a list of logically related radio buttons

      o only one radio box can be selected at a time

  print $query->p($query->radio_group(
                 -NAME => 'monty_hall',
                 -VALUES => [ "a new car", "a fabulous trip", "a donkey" ],
                 -DEFAULT => '-',
                 -LINEBREAK => 1,
   -LABELS => {
                    "a new car" => "door #1",
                    "a fabulous trip" => "door #2",
                    "a donkey" => "door #3",
},
                                  )
               ), "\n";


  -NAME : as for scrolling list
  -VALUES : as for scrolling list
  -DEFAULT : name of default button to turn on; specify non-existent
button to have none turned on
  -LINEBREAK : as for checkbox group (optional)
  -LABELS : as for scrolling list (optional)

    -- POPUP MENU

      o creates a popup menu the user can select from

      o only one menu item can be selected at a time

  print $query->p($query->popup_menu(
                 -NAME => 'pick',
                 -VALUES => [ "eenie", "meenie", "minie", "moe" ],
                 -DEFAULT => 'minie',
   )
  ), "\n";

  -NAME : as for scrolling list
  -VALUES : as for scrolling list
  -DEFAULT : name of default button to turn on; default is first
item
  -LABELS : [not used above] as for scrolling list (optional)
```

```
      -- SUBMIT

         o creates button used to submit the form

         o every form should have one

     -- RESET

         o creates a button to reset the form

         o every form should have one

    print p($query->submit( -NAME => "submit",
                            -VALUE => "go" ),
            $query->reset( -NAME => "clear" )
           ), "\n";

    -NAME : as for scrolling list (also used to label button)
    -VALUE : value returned to CGI script; overrides label given by
name on button [submit only] (optional)
```

To display current settings of named fields from a form:

```
sub display_values {
    my($query)=$_[0];

    my(@values, $key);

    print h2("Here are the current settings:"), "\n";
    print hr();

    foreach $key ($query->param) {

print b("$key ->");
        @values = $query->param($key);

        print join("; ", @values), br(), "\n";
    }

}
```

```
  # value for concurrent read lock
  # used when file is open for reading only
  $LOCK_SH = 1;
  # value for exclusive write lock
  # used when file is open for writing
  $LOCK_EX = 2;

  # we will talk about bail soon
  open(INFILE, "<$candidate_filename")
    or &bail("Cannot open file $candidate_filename for input : $!");
  # concurrent read lock after opening for read
  # anyone can still read; no one can write
  flock(INFILE, LOCK_SH)
    or &bail("Cannot lock file $candidate_filename for reading : $!");

  # do stuff

  # closing unlocks the file
  close(INFILE)
    or &bail("Error in closing file $candidate_filename for input : $!");


  open(OUTFILE, ">$candidate_filename")
    or &bail("Cannot open file $candidate_filename for output : $!");
  # exclusive write lock after opening for write
  # no one can write or read
  flock(OUTFILE, $LOCK_EX)
    or &bail("Cannot lock file $candidate_filename for writing : $!");

  # do stuff

  # closing unlocks the file
  close(OUTFILE)
    or &bail("Error in closing file $candidate_filename for input : $!");

# Note that this routine assumes
# that header() and start_html()
# have already been called
sub bail {

    my($error) = "@_";

    print h1("Unexpected Error"), "\n";
    print p($error), "\n";
    print p("Please contact", i("www\@club.org"));
    print end_html(), "\n";;
    die($error);
}
```