

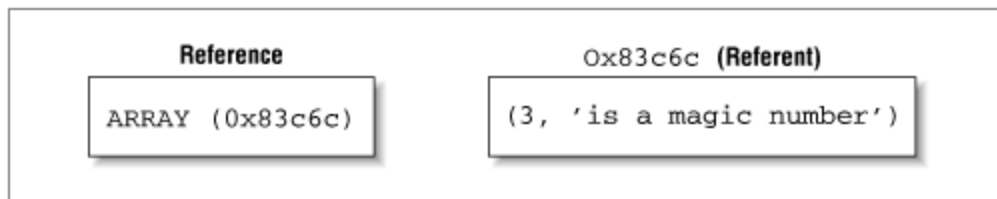
Intro to Computer Programming: week 4

Synopsis:

- Read HTML pages from Perl Cookbook on References
- Read HTML pages on the Perl CGI library
- Write Perl program that produces a web form html page

Details:

1. Read pages on Perl References, Arrays, and Hashes from Perl Cookbook
 - a. Below are subjects to read. The links below are to the full content of the section. The excerpts (parts) you need to read are in this document at the end in the same order as presented here:
 - i. Intro to references - http://docstore.mik.ua/orelly/perl/cookbook/ch11_01.htm
 - ii. Taking references to Arrays - http://docstore.mik.ua/orelly/perl/cookbook/ch11_02.htm
 - iii. Taking references to Hashes - http://docstore.mik.ua/orelly/perl/cookbook/ch11_04.htm
 - b. **If you need more review** on Arrays and Hashes, the intro sections on these data types in the Cookbook are very good:
 - i. Intro to arrays - http://docstore.mik.ua/orelly/perl/cookbook/ch04_01.htm#ch04-12050
 - ii. Intro to hashes - http://docstore.mik.ua/orelly/perl/cookbook/ch05_01.htm#ch05-22072
2. Read pages on Perl module library CGI
 - a. Below are subjects to read. The links below are to the full content of the section. The excerpts (parts) you need to read are in this document at the end in the same order as presented here:
 - i. Calling CGI.PM routines - <http://perldoc.perl.org/CGI.html#CALLING-CGI.PM-ROUTINES>
 - ii. Creating Fill-Out forms - <http://perldoc.perl.org/CGI.html#CREATING-FILL-OUT-FORMS%3a>
 - b. Each page below is about creating an HTML fill-out form element. Each page has code examples. If you have trouble understanding the section, copy and paste the example code into your program to see out it works
 - i. Creating a text field - <http://perldoc.perl.org/CGI.html#CREATING-A-TEXT-FIELD>
 - ii. Creating a popup menu - <http://perldoc.perl.org/CGI.html#CREATING-A-POPUP-MENU>
 - iii. Creating a radio group - <http://perldoc.perl.org/CGI.html#CREATING-A-RADIO-BUTTON-GROUP>
 - iv. Creating a submit button - <http://perldoc.perl.org/CGI.html#CREATING-A-SUBMIT-BUTTON>
3. Add code to the file test_CGI.cgi to make calls to the CGI library to create fill-out form elements.
 - a. In your cgi-bin dir there is a new file called test_CGI.cgi
 - b. Add an html anchor to your home page, index.html, that links to this code
 - i. Remember to use online sources at w3schools to determine how to write html:
 1. HTML reference: <http://www.w3schools.com/tags/default.asp>
 2. HTML anchors: http://www.w3schools.com/tags/tag_a.asp
 - c. The new file is a shell of a program that needs to be finished by adding form elements:
 - i. A text field
 - ii. A popup menu
 - iii. A radio button group
 - iv. A submit button



Referents in Perl are *typed*. This means you can't treat a reference to an array as though it were a reference to a hash, for example. Attempting to do so produces a runtime exception. No mechanism for type casting exists in Perl. This is considered a feature.

11.1. Taking References to Arrays

Problem

You need to manipulate an array by reference.

Solution

To get a reference to an array:

```
$aref = \@array;
$anon_array = [1, 3, 5, 7, 9];
$anon_copy = [ @array ];
@simplicit_creation = (2, 4, 6, 8, 10);
```

To dereference an array reference, precede it with an at sign (@):

```
push(@$anon_array, 11);
```

Or use a pointer arrow plus a bracketed subscript for a particular element:

```
$two = $simplicit_creation->[0];
```

To access a particular element of the array referenced by `$aref`, you could write `$$aref[4]`, but writing `$aref->[4]` is the same thing, and it is clearer.

11.3. Taking References to Hashes

Problem

You need to manipulate a hash by reference. This might be because it was passed into a function that way or because it's part of a larger data structure.

Solution

To get a hash reference:

```
$href = \%hash;
$anon_hash = { "key1" => "value1", "key2" => "value2", ... };
$anon_hash_copy = { %hash };
```

To dereference a hash reference:

```
%hash = %$href;
$value = $href->{$key};
@slice = @$href{$key1, $key2, $key3}; # note: no arrow!
@keys = keys %$href;
```

Excerpts to read on the Perl CGI library:

CALLING CGI.PM ROUTINES

Most CGI.pm routines accept several arguments, sometimes as many as 20 optional ones! To simplify this interface, all routines use a named argument calling style that looks like this:

```
1. print $q->header(-type=>'image/gif',-expires=>'+3d');
```

Each argument name is preceded by a dash. Neither case nor order matters in the argument list. -type, -Type, and -TYPE are all acceptable. In fact, only the first argument needs to begin with a dash. If a dash is present in the first argument, CGI.pm assumes dashes for the subsequent ones.

Several routines are commonly called with just one argument. In the case of these routines you can provide the single argument without an argument name. header() happens to be one of these routines. In this case, the single argument is the document type.

```
1. print $q->header('text/html');
```

Other such routines are documented below.

Sometimes named arguments expect a scalar, sometimes a reference to an array, and sometimes a reference to a hash.

A large number of routines in CGI.pm actually aren't specifically defined in the module, but are generated automatically as needed. These are the "HTML shortcuts," routines that generate HTML tags for use in dynamically-generated pages. HTML tags have both attributes (the attribute="value" pairs within the tag itself) and contents (the part between the opening and closing pairs.) To distinguish between attributes and contents, CGI.pm uses the convention of passing HTML attributes as a hash reference as the first argument, and the contents, if any, as any subsequent arguments. It works out like this:

1.	Code	Generated HTML
2.	----	-----
3.	h1()	<h1>
4.	h1('some','contents');	<h1>some contents</h1>
5.	h1({-align=>left});	<h1 align="LEFT">
6.	h1({-align=>left},'contents');	<h1
	align="LEFT">contents</h1>	

HTML tags are described in more detail later.

Many newcomers to CGI.pm are puzzled by the difference between the calling conventions for the HTML shortcuts, which require curly braces around the HTML tag attributes, and the calling conventions for other routines, which manage to generate attributes without the curly brackets. Don't be confused. As a convenience the curly braces are optional in all but the HTML shortcuts. If you like, you can use curly braces when calling any routine that takes named arguments. For example:

```
1. print $q->header( {-type=>'image/gif',-expires=>'+3d'}  
);
```

CREATING FILL-OUT FORMS:

General note The various form-creating methods all return strings to the caller, containing the tag or tags that will create the requested form element. You are responsible for actually printing out these strings. It's set up this way so that you can place formatting tags around the form elements.