ITE315 Module 3 Part D - Programming in Perl: Advanced Topics

Athens State University

Contents

1 References

References in Perl

- Early in learning about functions in C++, one is taught the difference between call by value and call by reference
  - By default, Perl does call by value
- Call by reference leads into the use cases for pointers in C++
  - Where you may need to refer to a value without making a copy

References in Perl

- The **reference** data type in Perl allows you to reference to a value without making a copy
- The `reference operator` “” is used to create a reference to another value
- The double scalar sigil ($$) dereferences a scalar reference
- In scalar contexts, it creates a single reference
- In list contexts, it creates a list of references

References in Perl

```
my $name = 'Larry';
my $nameRef = \$name;

sub reverseInPlace {
  my $nameRef = shift;
  $$nameRef = reverse $$nameRef;
}
my $name = "Blabby';
reverseInPlace(\$name);
say $name;
```

References in Perl

```
sub reverseInPlace{
  $_0 = reverse $_0;
}

my $name = 'allizocohC";
reverseInPlace( $name );
say $name
```

Note that the parameters in _ behaves as aliases to caller variables so they can be treated as references and modified in place.

It should be noted that this isn't good practice. Preventing people from doing this is why you assign parameter values to lexical local variables rather than directly manipulating the contents of the parameter array.

1.1 Array References

Array References

```
my @cards = qw(K Q J 10 9 8 7 6 5 4 3 2 A);
my $cardsRef = \@cards;
my $cardCount = @$cardRef;
my @cardCopy = @$cardsRef;
my $firstCard = $$cardsRef[0];
my $lastCard = $$cardsRef[-1];
my @highCards = @{ $cardsRef }[0 .. 2, -1];
```

Note the implications of the different sigils on the variables. We are taking a reference to an array, which results in a scalar of reference type. We apply the array dereference to scalar reference to get at the array value being referenced.

The curly brace dereferencing syntax can be used to slice an array reference using a range.

Anonymous Arrays

```
my $suitsRef = [qw( Monkeys Robots Dinos Cheese)];
my @meals = qw( soup sandwiches pizza);
my $sunday =  \@meals;
my $monday = \@meals;
push @meals, 'ice creame sundae';
```

In the first line, we assign a scalar to array of strings. The result is a reference to an anonymous array.

One must be careful. Think for moment: what value does `$sunday` and `$monday` refer to after the `push` statement?

Anonymous Arrays

```
my @meals = qw( soup sandwiches pizza);

my $sunday =  [ @meals ];
my $monday = [ @meals ];
push @meals, 'a really good pie';
```

By using anonymous arrays, the contents of the meals array is copied into the anonymous array and so you have three different arrays as result.

Hash References

```perl
 my %colors = (
       blue   => 'azul',
       gold   => 'dorado',
       red    => 'rojo',
       yellow => 'amarillo',
       purple => 'morado',
);
my $colors_ref = \%colors;
my @english_colors = keys %$colors_ref;
my @spanish_colors = values %$colors_ref;
sub translate_to_spanish {
  my $color = shift;
  return $colors_ref->{$color};
  # or return $$colors_ref{$color};
}
```

Anonymous Hashes

```perl

my $food_ref = {
  'birthday cake' => 'la torta de cumpleanos',
   candy => 'dulces',
   cupcake => 'bizcochito',
  'ice cream'=> 'helado',
};
my @cakes = ('birthday cake' 'cupcake');
my @translatedCakes = @{$food_ref }{ @colors }
```

We can do anonymous hashes in much the same manner as we do anonymous arrays. Do be careful about the use of parenthesizes and braces. Assignment of anonymous hash to a standard hash variable will result in Perl generating a really strange error about number of elements in the hash.

Function References

```perl
sub bakeACake { say 'I like cake.';};

my $cakeFuncRef = \&bakeACake

my $pieFuncRef = sub {say 'And I do like a pie';};

$cakeFuncRef->();
$pieFuncRef->();
```

Perl supports a computer science concept called **first-class functions**. From a practical standpoint, this means that you can treat functions as a data type as if they were an array or hash. There are lots of fancy CS-related things that you can do with function references. We'll leave it at that at this point in your careers (more in later courses).

2 Recursion

Recursion Is Your Friend

```
sub elem_exists {
  my ($item, @array) = @_;
  # break recursion with no elements to search
  return unless @array;
  # bias down with odd number of elements
  my $midpoint = int( (@array / 2) - 0.5 );
  my $miditem  = $array[ $midpoint ];
  # return true if found
  return 1 if $item  == $miditem;
  # return false with only one element
  return   if @array == 1;
  # split the array down and recurse
  return elem_exists(
          $item, @array[0 .. $midpoint]
         ) if $item < $miditem;
  # split the array and recurse
  return elem_exists(
           $item, @array[ $midpoint + 1 .. $#array ]
         );
}
```

3 Objects

Objects in Perl

- The core of Perl has minimal psupport for object-oriented programming
- The more common approach is to use one of the Object-oriented module distributions out of CPAN
  - Moose
  - Moo

Classes and Objects

- An object is some runtime entity that has state and behavior
- A class is a template that describes an object's state and behavior

Example of An Object in Moose

```
package Cat {
 use Moose;
 sub meow {
   my $self = shift;
   say 'Meow!';
 }
}

my $irriatingFuzzyAlarm = Cat->new;
$irriatingFuzzyAlarm->meow for 1 .. 3;
```

Every object can have its own distinct data. Methods which read or write the data of their invocants are instance methods; they depend on the presence of an appropriate invocant to work correctly. Methods (such as meow()) which do not access instance data are class methods. You may invoke class methods on classes and class and instance methods on instances, but you cannot invoke instance methods on classes.

Class methods are effectively namespaced global functions. Without access to instance data, they have few advantages over namespaced functions. Most OO code uses instance methods to read and write instance data.

Constructors, which create instances, are class methods. When you declare a Moose class, Moose provides a default constructor named new().

Example of An Object in Moose

```
package Cat {
 use Moose;
 sub meow {
   my $self = shift;
   say 'Meow!';
 }
 has 'name', is => 'ro', isa => "Str';
}
```

This says that the class `Cat` has an attribute (instance variable) called `name` that is a read-only attribute of type `Str`.

Moose automagically generates accessor methods that are functions that are the same name as the attribute (`name()` in this case).

```
for my $name (qw( Sylvester Garfield Bill)) {
   my $cat = Cat->new( name => $name);
   say "Created a cat named ", $cat->name;
}
```