Around the PERL in 11 days

Day 2
In retrospect

- Operators
- Scalars
- Variables

Comment….comment….comment

use long variable names

Use warnings

And make a scripts/bin directory
Arrays and List data

• A list is a bunch of scalar data.
• An array is a variable that stores a list.
• An array can store many many scalar data. (depends on available memory)
• Each element of an array is a scalar variable that stores an independent scalar value.
• A list is ordered data.

(1,2,3,4,5) #array of 5 values 1 to 5
(“Peter”,45,$name,$a+$b) #array of 4 values
() #empty list
#!/usr/local/bin/perl –w

#use strict;

#Storing some scalar data to an array and manipulating the array

$some_num=10;

@my_list_of_things = (“Peter”, 45, $some_num, $some_num * 2);

$length_mylist = @my_list_of_things ;   #gets the number of elements

print “The list is now”, @my_list_of_things ,”and has”, $length_mylist,”elements\n”;  

@my_list_of_things = (”front”,@my_list_of_things,”back”);

$last_index = $# my_list_of_things;       #gets the index of the last element

print “The list is now”, @my_list_of_things ,”and has”, $last_index,”elements\n”;
Arrays and list data: Array element access

- Now that we have data stored in an array, how do we access the individual data?

```perl
#!/usr/local/bin/perl -w

#use strict;

#Accessing data in a list

@my_list_of_things = ("Peter", 45, "Skywalker");
$first_element = $my_list_of_things[0];
$my_list_of_things[1]++;

$last_element = $my_list_of_things[-1];

print $first_element,"\t",@my_list_of_things,"\t",$last_element,"\n";
```
#!/usr/local/bin/perl

#Using functions push, pop, shift, unshift, reverse

@my_list = ("Peter", 45, "Skywalker");
push(@my_list,"Luke");  # @my_list=(@my_list,"Luke")
print "pushed \t",@my_list,"\n";
$name = pop(@my_list);  #removes last element of my_list
print "Popped \t$name, from \t",@my_list,"\n";
unshift(@my_list,$name);  # @my_list = ($name,@my_list);
print "Unshifted \t",@my_list,"\n";
$ano_name=shift(@my_list);  # ($ano_name,@my_list) = (@my_list)
print "Shifted \t$ano_name, from \t",@my_list,"\n";
@my_list = reverse(@my_list);
print "Reversed \t",@my_list,"\n";
Loop the loop: Control Structures

- We have an array with a list of numbers. Say we need to the same set of operations to every element of the array.
  
  for..
  
  foreach..

- We have to execute the same statements if a condition is met or another group of statements if not → Conditional statements
  
  if..elsif..else statements
  unless..else statements
  while loops
  until loops
  do..while loops
  do..until loops
Control Structures: for and foreach

for (initial exp; test_exp; reinitialize exp) {
    statement 1;
    statement 2;
}

#!/usr/local/bin/perl -w
#use strict;
#an example of the for loop. Using $i for the index.
@bases = qw(A T G C); #like @bases = ("A","T","G","C");
for($i=0;$i<=$#bases;$i++) {
    print "Element ",$i," of the array is ",$bases[$i],"\n";
}

#An example of the foreach loop.
foreach $element (@bases) {
    print "The element is ",$element,"\n";
}
Control Structures: if..else and unless

#!/usr/local/bin/perl -w
#Testing the if-else-elsif statement
print "How many amino acids are used in the synthesis of proteins?:";
$answer =<STDIN>;
chomp($answer);
if ($answer == 20) {
    print "That is the correct answer\n";
} elsif ($answer > 18 && $answer < 22) {
    print "Close..but no cigar\n";
} else {
    print "Waaay off mark.\n";
} #if-elsif-else
unless ($answer == 20) {
    print "Not the correct answer.\n";
}
Control Structures: Operators

- Comparison operators

<table>
<thead>
<tr>
<th>Comparison</th>
<th>String</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal</td>
<td>==</td>
<td>eq</td>
</tr>
<tr>
<td>Not Equal</td>
<td>!=</td>
<td>ne</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;</td>
<td>lt</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;</td>
<td>gt</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>&lt;=</td>
<td>le</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>&gt;=</td>
<td>ge</td>
</tr>
</tbody>
</table>

- Logical Operators

& &  - logical and

|| - logical or

! – logical not

xor – exclusive or
Control Structures: while, until

- For repeated iteration of a set of statements until the condition is met

```perl
#!/usr/local/bin/perl -w
# An example with the while loop
$some-number = 10;
while ($some-number > 7) {
    print "The number is now $some-number\n";
    $some-number--;
}
# Similar example with the until loop
print "The number after the while loop is $some-number\n";
until($some-number >10) {
    print "The number is now $some-number\n";
    $some-number++;}
print "The number after the until loop is $some-number\n";
```

BEWARE of ENDLESS LOOPS
Control Structures: do..while, do..until

- Similar to while/until loops ….but different

```perl
#!/usr/local/bin/perl -w

# An example with the do..while loop

$some-number = 10;

do {
    print "The number is now \$some-number\n";
    $some-number--;
} while ($some-number > 7) ;

#Similar example with the do..until loop

print "The number after the do..while loop is \$some-number\n";

do {
    print "The number is now \$some-number\n";
    $some-number++;
} until($some-number >10) ;

print "The number after the do..until loop is \$some-number\n";
```

**BEWARE of ENDLESS LOOPS**