Iterating in Perl: Loops

- Computers are great for doing repetitive tasks.
- All programming languages come with some way of iterating over some interval.
- These methods of iteration are called ‘loops’.
- Perl comes with a variety of loops, we will cover 4 of them:
  1. if statement and if-else statement
  2. while loop and do-while loop
  3. for loop
  4. foreach loop
if statement

Syntax:
if(conditional) {
    ...some code...
}

- if the conditional is ‘true’ then the body of the statement (what’s in between the curly braces) is executed.

#!/usr/bin/perl -w
$var1 = 1333;
if($var1 > 10) {
    print "$var1 is greater than 10\n";
}
exit;

Output?
1333 is greater than 10
if-else statement

Syntax:
if(conditional)
{
    ...some code...
}
else
{
    ...some different code...
}

-if the conditional is ‘true’ then execute the code within the first pair of curly braces.
- otherwise (else) execute the code in the next set of curly braces

#!/usr/bin/perl -w
$var1 = 13;
if($var1 > 100)
{
    print "$var1 is greater than 100\n";
}
else
{
    print "$var1 is less than 100\n";
}
exit;

Output?
13 is less than 100
Comparisons that are Allowed

- In perl you can compare numbers and strings within conditionals
- The comparison operators are slightly different for each one

- The most common comparison operators for strings:

<table>
<thead>
<tr>
<th>syntax</th>
<th>meaning</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>lt</td>
<td>Less than</td>
<td>“dog” lt “cat”</td>
</tr>
<tr>
<td>gt</td>
<td>Greater than</td>
<td>“dog” gt “cat”</td>
</tr>
<tr>
<td>le</td>
<td>Less than or equal to</td>
<td>“dog” le “cat”</td>
</tr>
<tr>
<td>ge</td>
<td>Greater than or equal to</td>
<td>“dog” ge “cat”</td>
</tr>
<tr>
<td>eq</td>
<td>Equal to</td>
<td>“cat” eq “cat”</td>
</tr>
<tr>
<td>ne</td>
<td>Not equal to</td>
<td>“cat” eq “Cat”</td>
</tr>
</tbody>
</table>

False! d > c
True! d > c
False! d > c
True! d > c
True! c = c
False! c ≠ C
- The most common comparison operators for numbers:

<table>
<thead>
<tr>
<th>syntax</th>
<th>meaning</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>120 &lt; 10</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>120 &gt; 10</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>120 &lt;= 10</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>120 &gt;= 10</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
<td>120 == 10</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
<td>120 != 10</td>
</tr>
</tbody>
</table>

Note: These numerical comparison operators work on numbers! They don’t apply to numerical characters as strings!

ex:  345 > 62    ← This is true
    “345” gt “62”  ← This is false!
elsif statements

-This type of conditional is a different rendition of the if-else statement

Syntax:

```python
if(conditional 1)
{
    ..code..
}
elsif(conditional 2)
{
    ..code..
}
elif(conditional 3)
{
    ..code..
}
else
{
    ..code..
}
```

-if ‘conditional 1’ is not true, then check to see if ‘conditional 2’ is true, else check the next conditional, etc…
Example of if loops in Action

```perl
#!/usr/bin/perl -w
$var1 = 11;
$var2 = 7;
$var3 = 4;
if($var1 > $var2)
{
    print "$var1 is greater than $var2\n";
}
elseif($var1 == $var3)
{
    print "$var1 is equal to $var3\n";
}
else
{
    print "$var1 is not equal to $var2 or $var3\n";
    print "$var1 is also less than the other variables\n";
}
exit;
```

Output?

```
11 is greater than 7
```
Is everyone still with me?

HI, THIS IS DILBERT.

HI, I'M A BUSINESS REPORTER FOR THE WALL TIMES POST GAZETTE.

I'M DOING A STORY ABOUT HOW DUMB... I MEAN DYNAMIC... YOUR NEW PRODUCT LINE IS.

THEN HE PROMISED NOT TO PRINT THE AMUSING NICKNAME I HAVE FOR OUR CEO.

YOU ARE SO DYNAMIC.

Logical Operators

- For programming, you need a way to evaluate whether or not something is true or false (1 or 0)

- The logical operators work for both strings and numbers.

Consider flipping a fair coin ONCE.

Let H = The coin comes up ‘Heads’

Let T = The coin comes up ‘Tails’

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
<th>Example</th>
<th>Value (1 or 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical ‘or’</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical ‘and’</td>
<td>H &amp;&amp; T</td>
<td>False! (0)</td>
</tr>
<tr>
<td>!</td>
<td>logical ‘not’</td>
<td>!(H) &amp;&amp; T</td>
<td>True! (1)</td>
</tr>
</tbody>
</table>
while loop

Syntax:

```
while(conditional)
{
  ..code block..
}
```

- while the ‘conditional’ is true, the body of the while loop will execute

```perl
#!/usr/bin/perl -w
$var1 = 0;
while( $var1 < 5)
{
  print "\\$var1 is now $var1\n";
  $var1++;
}
exit;
```

Output?

$var1 is now 0
$var1 is now 1
$var1 is now 2
$var1 is now 3
$var1 is now 4
do-while loop

Syntax:
```perl
do { ..code block.. }while(conditional);
```

- the body of the do-while loop will execute ONCE, then check the conditional and repeat if necessary
- * Note the semicolon!

```perl
#!/usr/bin/perl -w
$var1 = 0;
do {
    print "\$var1 is now \$var1\n";
    $var1++;
} while( $var1 < 5);
exit;
```

Output?
```
$var1 is now 0
$var1 is now 1
$var1 is now 2
$var1 is now 3
$var1 is now 4
```
for loop

Syntax:
for(statement; conditional test; iteration statement)
{
   ..code block..
}

-the for-loop is used primarily for iterating over a fixed interval
-the starting point is specified by ‘statement’
-the ‘conditional test’ checks to see if the ‘statement’ is still true
-the ‘iteration statement’ specifies how to change the ‘statement’
-so long as the ‘conditional test’ is true, the code block will be executed.
#!/usr/bin/perl -w

$var1 = 0;
for($var1=0; $var1 < 10; $var1++)
{
    print "$var1 now has the value: $var1\n";
}
exit;

Output?

$var1 now has the value: 0
$var1 now has the value: 1
$var1 now has the value: 2
$var1 now has the value: 3
$var1 now has the value: 4
$var1 now has the value: 5
$var1 now has the value: 6
$var1 now has the value: 7
$var1 now has the value: 8
$var1 now has the value: 9
**foreach loop**

Syntax:

```perl
foreach $variable (a range) {
  ..code block..
}
```

- `$variable` doesn’t have to be declared prior to the `foreach` loop
- the range has to have some finite size (the size of an array, the number of entries in a hash, the length of a string, a range of numbers, etc..)

Ex:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| ```perl
foreach $v (2..10)
``` | `$v` will take on the values 2,3,4,5…10 |
| ```perl
@ary1 = (2, 4, 9, 3);
foreach $v (@ary1)
``` | `$v` will take on the values 2, 4, 9, 3 |
Sample Program from Yesterday:

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

%myHash = (  
    name => "Damian",
    dept => "Basket Weaving",
    zipCode => 48108
);

print "Contents of hash: \n";
foreach $v (values %hash1)
{
    print "$v now contains: $v\n";
}
exit;
```

Example use of `values` keyword with hash

Output?

Contents of hash:
$v now contains: Damian
$v now contains: Basket Weaving
$v now contains: 48108
Final Notes on Loops

1. The ‘next’ command:

   Syntax: next;

   - When present within a loop, this command cause the program to skip to the next iteration.

```
#!/usr/bin/perl -w
@ary1 = (2, 4, 1, 0.5, -28.4, -3, 100.4, 88.5);
for($i=0; $i<scalar(@ary1); $i++)
{
    if( ($ary1[$i] > 0) )
    {
        if($ary1[$i] > 4)
        {
            next; ## skip numbers greater than 4
        }
        else
        {
            print "$ary1[$i] is: $ary1[$i]\n";
        }
    } ## end of outer if statement
} ## end of for loop
```

Output?

```
$ary1[0] is: 2
$ary1[1] is: 4
$ary1[2] is: 1
$ary1[3] is: 0.5
```