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:

syntax	meaning	example	
`lt`	Less than	“dog” lt “cat”	False! d > c
`gt`	Greater than	“dog” gt “cat”	True! d > c
`le`	Less than or equal to	“dog” le “cat”	False! d > c
`ge`	Greater than or equal to	“dog” ge “cat”	True! d > c
`eq`	Equal to	“cat” eq “cat”	True! c = c
`ne`	Not equal to	“cat” eq “Cat”	False! c ≠ C

- The most common comparison operators for numbers:

syntax	meaning	example
<	Less than	120 < 10
>	Greater than	120 > 10
<=	Less than or equal to	120 <= 10
>=	Greater than or equal to	120 >= 10
==	Equal to	120 == 10
!=	Not equal to	120 != 10

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 statments

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

Syntax:

```
if(conditional 1)
{
  ..code..
}
elsif(conditional 2)
{
  ..code..
}
elsif(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

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

$var1 = 11;
$var2 = 7;
$var3 = 4;

if($var1 > $var2)
{
  print "$var1 is greater than $var2\n";
}

elsif($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.
scottadams@aol.com
www.dilbert.com
I'M DOING A STORY ABOUT HOW DUMB... I MEAN DYNAMIC... YOUR NEW PRODUCT LINE IS.
7/16/03
© 2003 United Feature Syndicate, Inc.
THEN HE PROMISED NOT TO PRINT THE AMUSING NICKNAME I HAVE FOR OUR CEO.
YOU ARE SO DYNAMIC.

http://www.dilbert.com/comics/dilbert/archive/dilbert-20030716.html

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'

Syntax	Meaning	Example	Value (1 or 0)
`||`	logical '**`or`**'	H **`||`** T	True! (1)
`&&`	logical '**`and`**'	H **`&&`** T	False! (0)
`!`	logical '**`not`**'	**`!`**(H) **`&&`** T	True! (1)

while loop

Syntax:

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

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

```
#!/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:

```
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!

```
#!/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:

```
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:

foreach $v (2..10)	$v will take on the values 2,3,4,5…10

@ary1 = (2, 4, 9, 3);
foreach $v (@ary1)
	$v will take on the values 2, 4, 9, 3

Sample Program from Yesterday:

```
#!/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

exit;
```

Output?

```
$ary1[0] is: 2

$ary1[1] is: 4

$ary1[2] is: 1

$ary1[3] is: 0.5
```