

Loops and Conditionals

Logic and control are the next steps in learning a programming language. Loops let us repeat steps.

If, else, elsif

if(*CONDITIONAL*) { }

```
1 my $var = rand(10); # a RANDOM number generator
2 if( $var < 3 ) {
3   print "Variable ($var) is less than 3\n";
4 } elsif( $var <= 5 ) {
5   print "Variable ($var) is between 3 and 5\n";
6 } else {
7   print "Variable ($var) is > 5\n";
8 }
```

The Truth is out there

- Operators equals `==`, less than `<`, greater than `>`, less than or equal to `<=`, greater than or equal to `>=`
- `!` means take opposite of
- For strings equals is with the `eq`, less than is `lt`, and greater than `gt`
- Numbers, except for 0 are always true, undefined is always false
- `? :`, is a special operator for combining, you can use it to combine a test and performing an operation depending on if the test is true or false. Here we test if a value is bigger than 10, if so set it to 'yes' otherwise set it to 'no' ** `my $is_large = ($val > 10 ? 'yes' : 'no');`

One liners

If statements can be combined onto a single line and can include or not include parentheses.

```
1 my $i = 2;  
2 print "$i is even\n" if $i % 2 == 0;  
3  
4 $i++;  
5 print "$i is even\n" if($i % 2 == 0);
```

Logically speaking

- `True && True = True`
- `True && False = False`
- `True || True = True`
- `True || False = True`
- `! (True) = False`
- `! (False) = True`
- `! ($x && $y) = !$x || ! $y`
- `! ($x || $y) = !$x && ! $y`

if and unless

if will test if something is true and execute the code block. unless will test if something is false and then execute the code block.

```
1 if( $color eq 'red' || $color eq 'yellow' || $color eq 'orange' ) {  
2   print "The color is warm\n";  
3 } elsif( $color eq 'blue' || $color eq 'green' || $color eq 'purple' ) {  
4   print "The color is cool\n";  
5 }
```

Can also be written (partially) as

```
unless( $color eq 'red' && $color eq 'yellow' || $color eq 'orange' ) {  
  print "The color is cool\n";  
}
```

Some logic

Test if one number is larger than another

```
1 if( $num1 > $num3 ) {  
2   print "$num1 is larger\n";  
3 }
```

Test if two strings are equal

```
1 if( $str eq 'yellow') {  
2   print "found a yellow one!\n";  
3 }
```

Loop-de-Loop

while loops will execute a block of code as long as the conditional is true

until is also a way to loop, but will continue as long as the

```
1 my $n = 0;
2 while($n < 10) {
3   print "n is $n\n";
4   $n++;
5 }
6 $n = 0;
7 until($n > 10) {
8   print "n is $n\n";
9   $n++;
10 }
```


For looping

For loops, much like while loops. There are 3 components. The initialization, the test, and the iteration.

```
1 for( my $i = 0; $i < 10; $i++) {  
2   print "i is $i\n";  
3 }
```

The initialization is `my $i = 0`

The test is `$i < 10`

The iterator is `$i++`

This could also be written as a while loop.

```
1 my $i = 0;  
2 while($i < 10) {  
3   $i++;  
4 }
```

Loop control

Can Short-circuit a loop with `last`

```
1 my $lightning = 0;
2 my $johnny_five = 0;
3 while( $johnny_five < 1000 ) {
4   if( $lightning == 1 ) {
5     print "I'm fried!\n";
6     last;
7   } else {
8     print "I'm alive\n";
9   }
10  $lightning = int rand(10);
11 }
```

Continuation

Can also continue a loop with `next`, by stopping and going back to the top of the loop.

```
1 while( <DATA> ) {  
2     my $row = $_;  
3     chomp;  
4     if( substr($row,0,1) eq '#' ) {  
5         # this data has a comment, let's skip the lines starting with #  
6         next;  
7     }  
8 }
```

Scope

Scope defines the area in a program that variable is valid for. Inside the brackets ({}) any variable declared with them is valid for that scope.

```
1 my $toy = "Truck";
2 my $n = 0;
3 print "Toy is $toy before the if\n";
4 if( $n < 1 ) {
5     my $toy = "Transformer";
6     print "Toy is $toy inside the if\n";
7 my $toy2 = 'Train';
8 }
9 # $toy2 would not be available here
10 print "Toy is $toy outside the if\n";
```

If you do not declare the variable inside the loop, you can end up updating the value. Notice the missing 'my' inside the if block.

```
1 my $toy = "Truck";
2 my $n = 0;
3 print "Toy is $toy before the if\n";
4 if( $n < 1 ) {
5     $toy = "Transformer";
6     print "Toy is $toy inside the if\n";
7 }
8 print "Toy is $toy outside the if\n";
```

Parenthetically

In some cases you may have seen

```
1 print "hello\n";  
2 print("hello\n");
```

Both are valid, Perl will let you get away without parentheses in many cases. However if it is ambiguous it can cause problems. For example

```
use strict;  
use warnings;  
my $str = 'AB-CD';  
print join ",", split "-", $str, "\n";  
  
print "\n--\n";  
  
print join(" ", split( "-", $str)), "\n";
```

Combining concepts

Suppose you wanted to process a stream of digits and find where the '01' were. You could just use index to find it all the occurrences.

```
1 my $str = "110101210201010011110";  
2 my $ind = index($str,"01");  
3 while( $ind >= 0 ) {  
4   # when ind is -1 it means it got to the end of the string  
5   print substr($str,$ind,2); # print 2 digits  
6   $ind = index($str,"01",$ind+2);  
7 }
```

Note – this is not exactly how you would find specific codons in a DNA string because index is not going to respect the reading frame. You may need to do this with substr instead, inspecting a codon at a time.