Introduction to Perl Programming

Recitation

Week 4

Adapted from Dr. Jeff Solka's
BINF634 class slides

Topics

- Loops
  - while, foreach, for
- Split and join
- Input/Output
- Example

`while` loops for list processing

```
@genes = ("HOXB1", "ALPK1",
   "TP53");
while (scalar @genes > 0) {
    $gene = shift @genes;
    print "Processing gene
   $gene\n";
    # put processing code here
}
```

```
Processing gene HOXB1
Processing gene ALPK1
Processing gene TP53
```

```
@genes = ("HOXB1", "ALPK1", "TP53");
while (@genes) {
    $gene = shift @genes;
    print "Processing gene $gene\n";
    # put processing code here
}
$size = scalar @genes;
print "There are now $size genes in
   the list: @genes\n";
```

```
Processing gene HOXB1
Processing gene ALPK1
Processing gene TP53
There are now 0 genes in the list:
```

`foreach` loops for list processing

```
print "for loop to process all items from a list\n";
@genes = ("HOXB1", "ALPK1", "TP53");
foreach $gene (@genes) {
    print "Processing gene $gene\n";
    # put processing code here
}
$size = scalar @genes;
print "There are still $size genes in the list: @genes\n";
```

```
for loop to process all items from a list
Processing gene HOXB1
Processing gene ALPK1
Processing gene TP53
There are still 3 genes in the list: HOXB1 ALPK1 TP53
```

`for` loops for list processing

```
print "another for loop to process a list\n";
@genes = ("HOXB1", "ALPK1", "TP53");
$size = scalar @genes;
for (my $i = 0; $i < $size; $i++) {
    $gene = $genes[$i];
    print "Processing gene $gene\n";
    # put processing code here
}
$size = scalar @genes;
print "There are still $size genes in the list: @genes\n";
```

```
another for loop to process a list
Processing gene HOXB1
Processing gene ALPK1
Processing gene TP53
There are still 3 genes in the list: HOXB1 ALPK1 TP53
```

`join`: converting arrays to strings

`print "converting array to string\n";`
`@genes = ("HOXB1", "ALPK1", "TP53");`
`$string = join(" ", @genes);`
`print "String of genes: $string\n";`
`$size = length $string;`
`print "String has length: $size\n";`
 	
`converting array to string`
`String of genes: HOXB1 ALPK1 TP53`
`String has length: 16`
	`print "join with empty separator\n";`
	`@genes = ("HOXB1", "ALPK1", "TP53");`
	`$string = join("", @genes);`
	`print "String of genes: $string\n";`
	`$size = length $string;`
	`print "String has length: $size\n";`
	
	`join with empty separator`
	`String of genes: HOXB1ALPK1TP53`
	`String has length: 14`

join with newline separator

```
print "join with newline separator\n";
@genes = ("HOXB1", "ALPK1", "TP53");
$string = join "\n", @genes;
print "String of genes: $string\n";
$size = length $string;
print "String has length: $size\n\n";
```

```
join with newline separator
String of genes: HOXB1
ALPK1
TP53
String has length: 16
```

`split:` converting string to arrays

```
print "converting string to array\n";
$dna = "ATGCATTT";
@bases = split "", $dna;
print "dna = $dna\n";
$size = scalar @bases;
print "The list of $size bases: @bases\n\n";
```

```
converting string to array
dna = ATGCATTT
The list of 8 bases: A T G C A T T T
```

split: using separators

```
print "split on white space\n";
$string = "HOXB1 ALPK1    TP53";
@genes = split " ", $string;
print "$string\n@genes\n\n";


split on white space
HOXB1 ALPK1    TP53
HOXB1 ALPK1 TP53
```

```
print "split on 'P'\n";
$string = "HOXB1 ALPK1    TP53";
@genes = split "P", $string;
print "$string\n";
foreach $gene (@genes) {
    print "|$gene|\n";
}

split on 'P'
HOXB1 ALPK1    TP53
|HOXB1 AL|
|K1    T|
|53|
```

The @ARGV Array

Array @ARGV is the list of command line arguments for the program:

```
% myprogram.pl hello 73 abcdef
```

has effect of:

```
@ARGV = ("hello", 73, "abcdef");
```

Opening a File

Perl has two simple, built-in ways to open files:
    -- the shell way for convenience
    -- the C way for precision.

The shell way

```
% myprogram file1 file2 file3
% myprogram    <  inputfile
% myprogram    >  outputfile
% myprogram    >> outputfile
% myprogram    |  otherprogram
% otherprogram |  myprogram
```

Redirecting Standard Input and Output

INPUT PIPE: "<" redirects standard input to a file:

```
% readname.pl < infile
Hello, Joe Smith!  On your next birthday, you will be 43.
```

OUTPUT PIPE: ">" redirects standard output to a file:

```
% readname.pl < infile > outfile
% cat outfile
Hello, Joe Smith!  On your next birthday, you will be 43.
```

open Function

- The "open" function takes two arguments:
  - a filehandle, and
  - a single string comprising both what to open and how to open it.
- "open" returns true when it works, and when it fails, returns a false value and sets the special variable $! to reflect the system error.
- If the filehandle was previously opened, it will be implicitly closed first.

```
open INFO,    "<  datafile"  or die "can't open datafile: $!";
open RESULTS,">  runstats"  or die "can't open runstats: $!";
open LOG,     ">> logfile "  or die "can't open logfile:  $!";

Note: whitespace before or after file name is ignored.
```