Introduction to Perl Programming

Beginning Perl, Chap 2,3,

Review

- login/logout
- emacs editor
- running perl
- getting help
- literals

Literals: strings and numbers

```
% perl -e 'print 2 + 2; print "\n";'
% perl -e 'print "abc"; print "def\n";

# string "addition" (concatenation operator)
% perl -e 'print "one two" . " and three\n";

# mixing numbers and strings:
% perl -e 'print (2 * 2) . "\n";'
% perl -e 'print "2 + 2 = " . (2 + 2) . "\n";

# decimals and concatenations:
% perl -e 'print 2.3 + 2 . "\n";'
% perl -e 'print 2 . 3 + 2."\n";'
```

Challenge:

- Play with "string math" (what does '`x`' do with strings? '`/`' and '`-`' ?)
- Print a random number.
- Print 10 random numbers.
- Challenge: print the same randomly generated number 10 times.

Variables – scalars, lists, and hashes

- All programming languages (and programmable calculators) have places to store information–numbers or strings
- In 'C' or pascal, variables must be pre-declared, and look like any other "words" in the language: `int i, j; float mean;`
- Perl variables can be pre-declared (they must be pre-declared with `"use strict"`) but some programs do not pre-declare them
- Perl variables *must* beginwith `"$"`, `"@"`, or `"%"`
- Scalar variables begin with `'$'` – `$i, $j, $mean`
- a scalar variable can hold an integer `$i=1;` real `$pi=3.14159;` or string `$sequence="acgttcggaccctgat";`
- a computer program variable is a place to store something, not an arbitrary value like in mathematics.

Lists (arrays, vectors)

- Most computer programs manipulate more complex (or structured) data – lists or vectors, matrices (2D arrays), etc.
- Perl provides a 1-d list or vector data type; more complex 2d or 3d data are more complex in perl (in contrast with 'C', pascal, or Fortran).
- Perl list variables (arrays) begin with `'@'` : `@nt=('a','c','g','t')`
- lists (arrays) have individual elements, and a size (or length)
- a list (array) element is always a scalar, and is referred to as: `$nt[0]`, `$nt[1]` , etc. Scalars always begin with `'$'`. `'@'` things are always lists (never scalar list elements).
- Lists (arrays) begin with element `0 $list[0]`
- The index of the last element is `$#list;` (the length of the list is `$#list+1`)

Initializing arrays and manipulating lists (arrays)

- `@list=(1,2,3,4,5);`
- `@list=(1,3.14159,"Pi");`
- `@nt=('a','c','g','t');`
- `@nt=qw( a c g t );`
- `@purine=qw(a g); @pyrimidine=qw(c t);`
- `@nt = (@purine, @pyrimidine)=`
  `('a','g','c','t');` (Lists "flatten")
- `$a = 'a'; $c='c'…; @nt = ( $a, $c, $g, $t );`
- `($a,$c,$g,$t) = @nt;`
- `@lines = split("\n",$lots_of_lines);`
- `@words = split(" ",$lots_of_words);`
- `@letters = split("",$string);`
- `@nt = split("","acgt");` but usually its `split(//,"acgt");`

Hashes: "associative arrays"

A hash is like a dictionary; it's a list of unique "words" (called "keys"), each of which has a "definition" (called a "value").

Hashes are initialized like arrays:

```
%hash = ('key','value','one', 1, 'two', 2,
2,'two',"William Pearson", "924-2818");
```

Perl has a special "comma" called "right arrow" that is particularly useful for creating hashes; it automatically "stringifies" the scalar to its left, and emphasizes the associativity:

```
%hash = ( key=>'value', one => 1, two => 2,
2=>'two');
```

Using Hashes

Values in a hash are accessed via its key; this is why hashes are often called “lookup tables”:

```
$value = $hash{‘key’};
$num = $hash{‘one’};
$num2 = $hash{two}; # no quotes necessary
$hash{three} = 3;   $hash{“William Pearson”};
```

Arrays of keys and values can be obtained independently via the ‘keys’ and ‘values’ functions:

```
@keys = keys %hash;
@vals = values %hash;
```

Hash “slices”

You can get more than one value out of a hash in one statement:

```
($num, $num2) = @hash{‘one’, ‘two’};
($num, $num2) = @hash{qw(one two)};
```

You can also modify the hash simultaneously:

```
@hash{qw(three four five)} = (3, 4, 5);
```

Iterating over hashes

Two ways to iterate over all the pairs in a hash:

```
for my $key (sort keys %hash) {
    my $val = $hash{$key};
    # do something with $key, $val
}
```

Or:

```
while(my ($key, $val) = each %hash) {
    # do something with $key, $val
}
```

Exercises

1. Generate a random DNA sequence (at least 100 nt)
2. Generate a random protein sequence ≥ 200 aa
3. Write a program to translate your random DNA sequence into protein (make certain it works if length(DNA)%3 != 0)