Perl For Beginners

What is PERL?

- Practical Extraction Reporting Language
- General-purpose programming language
- Creation of Larry Wall 1987
- Maintained by a community of developers
- Free/Open Source
  - www.cpan.org

Why use Perl?

- Perl is fast, especially at common tasks in biology: file manipulation and pattern matching
- Good at manipulating large data sets or performing the same task repeatedly
- CGI module gives simple interface for delivering dynamic web pages
- DBI modules provide database-independent interface for Perl
- Powerful easy-to-use modules for network programming (Web, E-Mail, FTP, etc.)
- TMTOWTDI

Overview

- Scalars, scalar variables and operations
- Control blocks and conditions
- Array, array variables and array operations
- Tips and resources

The Basics

- Text file using ordinary text editor (nedit, emacs)
- Comments begin with a pound-sign (#)
- Statements end with semi-colon (;)
- White space independent
- Case sensitive
- Variables need not be declared or “typed”

Scalars

- Represent a single piece of data
- Can represent string or numeric
  - 2, 3.1456, 1e-27, “ATC”, ‘NM_000327’
- Scalar Variables
  - Variable names consist of a dollar sign ($) followed by a letter or underscore then followed by zero or more letters, digits or underscores
    - $name, $old_name
  - Used to hold results of calculations, constants, input from keyboard, files, etc
    - $acc_number = “NM_000327”;

Numeric Operators

- Addition (+), subtraction (-), multiplication (*), division (/), modulus(%), exponentiation (**)
  - $a=1;
  - $b=2;
  - $c=$a + $b; # $c equals 3
  - $d=$c**2; # $d equals $c to the power of 2 which is 9
  - $e=$d%2; # $e equals the remainder of 9/2 which is 1

Numeric Comparison Operators

- == (equality), != (inequality), >(greater than), >=(greater than or equal to), < (less than), <= (less than or equal to)
  - $a=1; $b=2;
  - $a==$b # false
  - $a != $b # true
  - $b >= $a # true

String Operators

- . (concatenation), eq (equality), ne(inequality)
  - $a = “Hello ”;
  - $b = $a . “World”; # $b equals “Hello World”
  - $a eq “Hello”; # evaluates to true
  - $a ne “World”; # also evaluates to true

Variable Interpolation

- Variables are interpolated within double quotes but not within single quotes
- $a = ‘student’;
- “hello $a”; # evaluates to “hello student”
- ‘hello $a’; # evaluates to “hello $a”
- New lines (\n), tabs (\t) and other special characters interpolated within double quotes
  - print “hello\tstudent\twelcome\tto\Boston\n”;
  - prints tabs between each word and a trailing new line

Statements Blocks

- Curly Braces surrounding multiple statements
- # this is a naked block
- {
  - Statement 1;
  - Statement 2;
- }
- Naked block has no effect on program flow
- Blocks are typically part of a larger construct
- Types: while, for, foreach, if/else

if/elsif/else

```
if (test_expression)
{
Statement 1;
Statement 2;
}
elsif (test_expression2)
{
        Statement 3;
}
else
{
        Statement 4;
}
```

- Statement 1 and 2 are executed if test_expression is true, Statement 3 is excuted if test_expression2 is true, otherwise statement 4 is executed

More on if/else

- Braces are required (unlike other languages)
- else is optional
- “unless” can be used instead of “if” which reverses the test
- If more than two conditions exist use “elsif”

Arrays

- List of scalars
- Can store heterogeneous information
- No space allocation. Expands as necessary
- Ordered sequentially and indexed (start at 0)
- Variable names start with @
  - @bases = (“A”,”T”,”G”,”C”);
  - 0 1 2 3 # index

Array Assignments

@a = (7.34,“coffee”, “tea”, 343);

- qw - use white space to separate elements
  - @a = qw (7.34 coffee tea 343);
- Can be made up of scalar and array variables
  - @bases=qw(A T G C);
  - $a = “N”;
  - @legal_bases = (@bases, $a, “X”) # ATGCNX

Accessing Array Elements

- An array element can be retrieved by accessing its index
- @bases = qw (A C T G);
  - $third_base = $bases[2]; # $third_base equals T
- Elements can also be modified this way
  - $bases[3]='X'; # @bases now (A C T X)
- Negative subscripts count backward
  - $bases[-1]; #refers to last element X
- @bases and $bases are completely different

Filehandles

- To read from or write to a file in Perl, it first needs to be opened. In general, open (filehandle, filename);
- Filehandles can serve at least three purposes:
- open (IN, $file); # Open for input
- open (OUT, “>$file”); # Open for output
- open (OUT, “>>$file”); #Open for appending
- Then, get data all at once @lines =<IN>;
- Or one line at a time
- while (<IN>){
  - $line = $_ # do stuff with this line
  - Print OUT “This line: $line”;

Perl Functions

- Functions for scalars or strings
  - chomp, chop, chr, crypt, hex, index, lc, lcfirst, length, oct, ord, pack, q/STRING/, qq/STRING/, reverse, rindex, sprintf, substr, tr///, uc, ucfirst, y///
- Regular expressions and pattern matching
  - m//, pos, quotemeta, s///, split, study, qr//
- Numeric functions
  - abs, atan2, cos, exp, hex, int, log, oct, rand, sin, sqrt, srand
- Functions for real @ARRAYs
  - pop, push, shift, splice, unshift
- Functions for list data
  - grep, join, map, qw/STRING/, reverse, sort, unpack
- Input and output functions
  - binmode, close, closedir, dbmclose, dbmopen, die, eof, fileno, flock, format, getc, print, printf, read, readdir, rewinddir, seek, seekdir, select, syscall, sysread, sysseek, syswrite, tell, telldir, truncate, warn, write

Chop and Chomp

Chop

- Removes the last character of a string

- $a = “testing 123”;
- chop $a;
- # $a now equals “testing 12”

- Chomp
- Removes the last character of a string only if it is a newline (/n)

- $b = “this is a test\n”;
- chomp $b;
- # $b now equals “this is a test”

Pop and Push

Pop

- Removes and returns the last value of the array
- @bases = qw(A C T G);
- $z = pop @bases;
- #$z is G and @bases is (A C T)

Push

- Adds elements to the end of the array
- @a = (4, 5, 6, 7);
- push @a, 8;
- #@a is now (4, 5, 6, 7, 8)

Shift and Unshift

Shift

- Removes and returns the first element off the array
- @n = (9,8,7,6);
- $a = shift @n;
- # $a equals 9, @n = (8,7,6);

- Unshift

- Adds elements to the beginning of an array
- @y = (25, 26, 27);
- Unshift @y, 24;
- # @y becomes (24, 25, 26, 27)

Resources

- http://learn.perl.org/
- http://www.oreilly.com/
- BaRC Library
- Unix-Perl course Spring 2005 http://jura.wi.mit.edu/bio/education/bioinfo2005/unix-perl/
- Perl Library http://iona.wi.mit.edu/bio/bioinfo/scripts/