Lecture #17 – Perl Part I

- Background

  Written by Larry Wall and was first posted to comp.sources in late 1987
  Perl stands for Practical Extraction of Report Language (among other things)
  Wall quips it also stands for “Pathologically Eclectic Rubbish Lister”
  Originally designed for text processing language
  Grown into general purpose programming language
  Generally a simple language, but it is very feature rich (lot to learn)

  Version 4 out in 1987, and was considered a stable, standard UNIX prog language
  Version 5 came out later that year, and is now a good general programming language

  “Perl is designed to make the easy jobs easy, without making the hard jobs impossible”
  "There's more than one way to do it"
  Rich with non-orthogonality and shortcuts

- A First Perl Program

```
#!/usr/bin/perl –w

use strict;                          # force variables to be assigned before use
print "What is your username?  ";     # output a question to user
my $username;                        # "declare" a variable
$username = <STDIN>;                 # assigned value to variable
chomp($username);                    # cut off newline character
print "Hello, $username.\n";          # print out a greeting
```

- Program Structure

  Syntax is fairly forgiving

  1. whitespace - only required between items that would be otherwise confused as one item.

     Includes spaces, tabs, newlines, and comments

  2. semicolon - all simple statements end with a semicolon (;)

     compound statements are enclosed in braces {} and do not need one
     also last statement in a block does not need a semicolon

  3. declarations - only required for subroutines and report formats

  4. comments - follow # just like most other scripting languages

- Data Types

1. Scalar = simple variable (preceded with $); include number, string, or references

   Ex: The string "foobar" (i.e. $answer = foobar;)
   The number 3 (i.e. $number = 3;)
   The number 3.5 (i.e. $number = 3.5;)

   Generally no need to specify if something is string, float, integer, etc since Perl can usually figure it out.

   Strings:
   Sequence of characters usually delimited by ' or "
   Memory does not need to be allocated; they grow and shrink as necessary

   Numbers:
   Stored as signed integers or double precision floating point

   Ex:
```
$sign = “I love my $pet”;
$cost = ‘It costs $100’;
$cwd = `pwd`;
$exit = system(“vi $file”);
```

2. Arrays = ordered list of scalar with index starting at 0 (start with @)

```
();                                  # the empty list
qw//;                               # another empty list
("a", "b", "c", 1, 2, 3)            # a list with six elements
qw/Hello World  How are you/        # a list with five elements

#  assign 3 values to array home

@home = (“couch”, “chair”, “table”);

OR

$home[0] = “couch”;
$home[1] = “chair”;
$home[2] = “table”;

#  assign 3 elements of home into scalars

($potato, $lift, $tennis, $pipe) = @home;
```

3. Hashes = unordered sets of key/value pairs with keys as subscripts (start with %); similar to associative arrays

```
%longday = (“Sun”, “Sunday”, “Mon”, “Monday”);

OR

%longday = (
        “Sun” => “Sunday”,
        “Mon” => “Monday”
);
```

- Variables

Start with $, @, or % to identify data type
begin with letter or underscore
can contain letters, underscore, and digits (max length 255)

Ex:

```
$age = 26;
@date = (8, 24, 70)

%fruit = ('apples', 3, 'oranges' , 6)

OR

%fruit = (
        apples => 3,
        oranges => 6
)
```

Ex:

```
my @stuff = qw/a  b  c/;                        # three elements
my @things = (1, 2, 3, 4);                      # four
my $oneThing = "all alone";
my @allOfIt = (@stuff, $oneThing, @things);     # eight elements
```

Ex:

```
my @someStuff = qw/Hello and welcome/;          # three elements

$#someStuff = 0;                                # Now just ("Hello")
$someStuff[1] = "Joe";                          # ("Hello", "Joe")
$#someStuff = -1;                               # Now empty
@someStuff = ();                                # Same as last line
```

Ex:

```
my @stuff = qw/everybody wants a rock/
my @rock = @stuff[1 .. $#stuff];          # qw/wants a rock/
my @want = @stuff[0 .. 1];                # qw/everybody wants/
@rock = @stuff[0, $#stuff];               # qw/everybody rock/
```

Note: It is possible to use arrays as stacks or queues with functions push, pop, unshift

namespace for $, @, and % are separate; therefore, $foo and @foo are different variables
variable names are case sensitive

Several implicit conversions take place…

Ex:

```
$b = @c;          (num of elements in array c is assigned to b)
$a = (2, 4, 5, 8);    (a is assigned 8)
$a = [2, 4, 5, 8];    (a is a reference to an unnamed array)
```

Ex:

```
my @things = qw/a few of my favorite/;
my $count = @things;                      # will be 5
my @moreThings = @things;                 # same as @things
```

Ex:

```
my @saying = qw/these are a few of my favorite/;

my $statement = "@saying things.\n";
# "these are a few of my favorite things.\n"

my $stuff = "@saying[0 .. 1] @saying[$#saying – 1, $#saying] things.\n"
# "these are my favorite things.\n"
```

Ex:

```
$wife{“Adam”} = “Eve”;
# hash element is assigned a value of “Eve”

$wife{“Jacob”} = [“Leah”, “Rachel”, “Bilhah”, “Zil;pah”];
#  assign a reference to an unnamed array to the hash element $wife{“Jacob”}
```

- Control Structures

- if / unless

  General Syntax:

```
if (exp) { block } else { block }
unless (exp) { block } else { block }

if (exp1) {
        block1
}
elsif (exp2) {
        block2
}
else {
        block3
}
```

  Ex:

```
if ($debug > 0) {
        print "Debug:  Danger Will Robinson\n";
}

if ($city eq "New York") {
        print "New York is northeast of Washington, D.C.\n";
}
elsif ($city eq "Chicago") {
        print "Chicago is northwest of Washington, D.C.\n";
}
else {
        print "Don't know where $city is \n";
}

unless ($destination eq $home) {
        print "I'm not going home\n";
}
```

- while / until

  General Syntax:

  ```
  while (exp) { block }
  until (exp) { block }
  ```

  Ex:

  ```
  # Read each line from file handle INFILE and write it to file handle OUTFILE

  while (<INFILE>)                    #  could use $line = <INFILE> instead
  {
          print OUTFILE, "$_\n"
  }
  ```

  There is an optional continue block executed before loop iteration
  even if main portion of block is skipped (with continue, goto, etc)

  ```
  $i = 1;

  while ($i < 10)
  {
  }
  continue
  {
          $i++;
  }
  ```

- do while / until

  ```
  do { block } while (expression);
  do { block } until (expression);
  ```

- for / foreach

  for loop basically same as C for loop

  ```
  for ($i = 1; $i < 10; $i++)
  {
          print “$i\n”;
  }
  ```

```
foreach var (list)
{
        block
}
```

foreach loop can also take a continue block

Ex:

```
my @collection = qw/hat shoes shirts shorts/;

foreach my $item (@collection)
{
        print "$item\n";
}
```

Ex:

```
foreach $key (sort keys %hash) {

}
```

Ex: next and last

```
foreach $user (@users) {
        if ($user eq “root” or $user eq “lp” {
                Next;
        }

        if ($user eq “special”) {
                print “Found the special account\n”;
                #  processing here
                last;
        }
}
```

- Modifiers

Any simple statement can be followed by a modifier which is a shortcut to a compound control structure.

```
stmt if EXPR;
stmt unless EXPR;
stmt while EXPR;
stmt until EXPR;
```

Ex:

```
$i = $num if ($num < 50);
$lines++ while <FILE>;
print "$_\n" until /The End/;
```

- Loop Control

can name loops with a label

```
LINE:  while (<SCRIPT>)
       {
              print;
              next LINE if /^#/;
       }
```

last <label> is basically same as C break
next <label> is basically same as C continue
redo <label> restarts loop block without evaluating conditional
expression or executing the continue block

- Special Variables

$_ = default input and pattern matching string

```
foreach ('hickory', 'dickory', 'doc')
{
       print;
}
```

Perl assumes $_ if not specified in
print & unlink
pattern match ops m//, s///, tr///
when used without = ~ op
default iterator var in foreach
implicit iterator in grep / map
default place to put line input (i.e. while (<FILE>))