CPS160
Perl Tutorial 1

Abrita Chakravarty
Department of Computer Science
Duke University

Material adapted after Pallavi Pratapa, Jason Stajich, and Raluca Gordan (former CPS160 TAs)

Outline

- Variables
  - Scalars: strings and numbers
  - Arrays
  - Hashes
- Conditionals and loops
- File handling
- References

First Perl Program

```
use strict;
use warnings;
use cps160lib;

my @numbers = (12, 34, 54, 2, 423, -14);

my $minval = min(@numbers);# find minimum in the array
my $maxval = max(@numbers);# find maximum in the array

print "The array of numbers is (@numbers).\n";
print "The minimum value is: $minval.\n";

exit;
```

Coding Practices

- Put `use strict;` and `use warnings;` at the beginning of your programs
  - `strict` requires variables to be declared before use
    - `my $var = 3;`
    - `my` prevents context problems with multiple declarations
  - `warnings` provides warning messages to help debug
- Put `use cps160lib;` for helper code in cps160
- **ALWAYS** comment your code (# starts a comment)

First Perl Program

```
use strict;
use warnings;
use cps160lib;

my @numbers = (12, 34, 54, 2, 423, -14);

my $minval = min(@numbers);# find minimum in the array
my $maxval = max(@numbers);# find maximum in the array

print "The array of numbers is (@numbers).\n";
print "The minimum value is: $minval.\n";

exit;
```

Perl Variables

- Scalar: `$var`
  - Strings:
    - `$var1 = 'literal';`
    - `$var2 = "interpolate variable";`
  - Numbers:
    - `$num1 = 214; $num2 = 0.002; $num3 = 2e-3;`
- Array: `@array`
- Associative array or hash: `%hash`
- Note: case sensitivity; `$var` different from `$Var`

Strings

- Quotes
  - single quotes (exactly as typed)
    - `'literally $3.00 $var'`
  - double quotes (substitutes values for variables)
    - `"value = $var\n"`
  - back quotes (execute commands in shell)
    - `` `dir`, `ls` ``
- Formatting
  - `$str = sprintf("%s %.2f", 'formatting',9);`

String Functions

- Uppercase, lowercase
  - `$CAPS = uc($word);`
  - `$small = lc($WORD);`
- Equality
  - `if( $a eq $b ) {...}`
- Length
  - `$len = length($str);`
- Substring
  - `$partOfString = substr($str, 3, 5);`

String Functions (Contd.)

- Index of a substring
  - `$index = index($str, "b");`
- Concatenate two strings
  - `$sum = $part1 . $part2;`
  - (do not use "+" to concatenate strings!)
- Reverse a string
  - `$revstr = reverse($str);`

Manipulating Strings

- Append to a string:
  - `$str .= "stuff to add at the end";`
- Remove the end if it is whitespace:
  - `chomp($str);`
  - Also removes new-line character (enter/return)
- Concatenate text that crosses several lines:
  - ```
    $str = "This is a long sentence that ”
           ."continues onto more than one line."
    ```
    ```

Numeric Variables and Operators

- Integers
  - `$num = 2030`
- Floating points
  - `$float = 0.400323`
- Scientific notation
  - `$evalue = 1.32e-23`
- Basic Arithmetic
  - `+,-,/,*`
- Power **
- `log()` (base e)
- Increment/decrement
  - `$var++, $var--`
  - `++$var, --$var`
- Assignment
  - `$a += 3;`
  - `$num /= $num2;`

Operator Precedence and Associativity

- **Precedence**: some operators evaluated before others.
  - `2 + 4 * 5 = 2 + 20 = 22`
  - **not** `(2 + 4) * 5 = 6 * 5 = 30`
- **Associativity**: order of evaluating multiple occurrences of same operator
  - Left associativity:
    - `72 / 12 / 3 = (72 / 12) / 3 = 6 / 3 = 2`
    - **not** `72 / (12 / 3) = 72 / 4 = 18`
  - Right associativity:
    - `2 ** 3 ** 4 = 2 ** (3 ** 4) = 2 ** 81 = 2.41e+24`
    - **not** `(2 ** 3) ** 4 = 8 ** 4 = 4096`
- **Examples: ex1a.pl, ex1b.pl**

Arrays and Lists

- Array variables start with `@`
- An array holds a list of items; we can mix types (numbers, strings, objects, even other arrays).
  - `@array = ('banana', 'apple', 200);`
- Arrays are automatically resized .
  - `$array[1000] = '1';`
- Get the 3rd item in the array
  - when referencing an element of an array, you use `$`, not `@`
  - `$item = $array[2];`
- Get a slice of an array
  - `@newarray = @array[0..3];`

Array Functions

- Remove last item
  - `$item = pop @array;`
- Add to the end
  - `push @array,$item;`
- Remove 1st item
  - `$item = shift @array;`
- Add to the front
  - `unshift @array,$item;`
- Reverse order of elements
  - `@revarray = reverse @array;`

Array Size

- Use **`scalar`**, not **`length`** to determine the number of items in an array
  - `scalar()` returns the size of an array:
  - `$array_size = scalar(@array);`
  - `length()` returns the number of characters in a string:
  - `$string_length = length($string);`

From String to Array and Back

- Turn a string into an array; Use a delimiter
  - `@words = split(" ","the quick brown fox”)`
- Turn and array into a string
  - `$str = join(" ", @words)`
- Make an array of strings quickly
  - `@spring_months = qw(Mar Apr May);`

Context in Perl

- Perl tries to be clever
- Will do different things depending on context, so you must be careful !
- `$length = @array;`
- `($first_element) = @array;`
- `($first, $second) = @array;`

- **Example: ex2.pl, ex3.pl**

Associative Arrays or Hashes

- Key => value pairs
- Similar to arrays, except that can use meaningful strings to index, rather than just integers
- No implicit order
- %hash

Manipulating Hashes

- Add a key, value pair
  - `$hash{'Y'} = 'You';`
- Initialize at once
  - `%hash = ('Y' => 'You', 'M' => 'Me');`
- Remove a value
  - `delete $hash{'A'};`
- Test if a key has been set
  - `if( exists $hash{'P'} ) {}`
- Test if a key has been set and has been defined
  - `if( defined $hash{'P'} ) {}`

Hash Functions

- Get the keys
  - `my @keys = keys %hash;`
- Get the values
  - `my @values = values %hash;`
- Loop through both
  - `while(my($key,$value) = each %hash) {…}`
- **Example: ex4.pl**

Conditionals and Loops

- **if**(*something*) { *something is true* }
- **elsif**(*something else*) { *something else is true* }
- **else**{ *neither something nor something else are true* }

- **while**(boolean){}
- **until**(!boolean){}
- **for**(*initialize; booleantest; incr/decr*){}
- **foreach** $var (@list){}

Boolean operators

- Equivalent
  - numeric: `==`
  - characters: `eq`
- and (`&&, and`), or (`||, or`), not (`!, not`)
- `unless` is `!if`
  - `if (!$a ){}` is the same as
  - `unless ($a){}`

Input / Output Streams

- STDOUT, STDERR output streams
- STDOUT is the default if no output stream is specified
  - `print "an error occurred\n";`
  - `print STDERR "an error occurred\n";`
- STDIN input stream

```
$filename = <STDIN>;
chomp($filename);
print "Filename: ";
```

Input Filehandles

- Open a file for reading; use “< ”
  - `open(IN, "<$filename”)`
- Example:

```
open(IN, “<$filename”) || die("error\n");
while (my $line = <IN>)
{ print "line is $line“; }
close(IN);
```

- Filehandles used with the Diamond operator < > when assigning to a variable

Output Filehandles

- Use ">" to create a new file (or overwrite text in an existing file) , and ">>" to append to end of a file.

```
open(FH, ">filename") || die($!);
print FH "Writing a line to file\n";
close(FH);
```

- `$!` contains a human readable error message.

Program Inputs in CPS160

- STDIN: `my $filename = <STDIN>;`
- FILE: `open(IN, "<$filename")`
  - Input files: *.fasta, *.txt
  - Read in a line and parse it

```
open(IN, $filename) || die("error\n");
while (my $line = <IN>)
{
   my @wordArray = split " " $line);
}
close(IN);
```

- Helper code provided: `cps160lib.pm(getfasta), tidy.pl`
- **Example: ex5.pl**

References

- Represent memory addresses of variables (pointers)
- References:
  - leading `\`
  - `[]` for arrays
  - `{}` for hashes
- Note: These are used for slightly different things
  - `$ref = \$var;`
  - `$aref= \@array; $arrayref = [@anonArray];`
  - `$href= \%hash;  $hashref = {%anonHash};`

References with []

- It's ok to write:
  - `@array = ($a, $b, $c);`
  - `$goodref= \@array;`
- But
  - `$badref= \($a, $b, $c);`

  is the same thing as
  - `$badref= (\$a, \$b, \$c);`
- A reference to an anonymous array:
  - `$goodref= [$a, $b, $c];`

Data Structures

- References can be used to create complicated data structures.
- A simple example: a matrix is an array of references to arrays:

```
my @matrix = ([67, 78, 98],
              [99, 95, 82]);
```

- Arrays of hashes, hashes of arrays, hashes of hashes …
- Note: references cannot be used as hash keys!

De-referencing

- Get the data from the reference
- Have to know what was stored (scalar, array, or hash)
- Get back the scalar, array or hash
  - `$$scalarref, @$arrayref, %$hashref`
- Use the reference to get an array or hash element
  - `$arrayref->[2]; $hashref->{“key2”}`
- **Example: ex6.pl**

Further Reading

- http://www.cs.duke.edu/courses/fall10/cps160/resources.html
- Online resources
  - http://perldoc.perl.org/
  - http://www.perl.org/books/beginning-perl/
- Textbook
  - Beginning Perl, Second Edition, James Lee