Introduction to Perl

How to run perl

- Perl is an interpreted language. This means you run it through an interpreter, not a compiler.
- Your program/script must first tell the system where the interpreter is located
- This is done via the "shebang"
- `#!/usr/local/bin/perl`
  - mildly different from the book

One more step…

- Must tell the OS that this is an executable file.
- Use chmod (see intro to unix slides)
- Usually only need to give yourself execute permissions.
- Once it's executable, type the filename at a prompt, and it runs.

Very basic I/O

- simple introduction to reading/writing from keyboard/terminal.
- More advanced (ie, File) I/O will come around in a couple weeks.
- This will be just enough to allow us to do some examples, if necessary.

Output to terminal

- the `print` statement.
- Takes 0 or more arguments.
- First (optional) argument is the filehandle.
  - if omitted, prints to STDOUT.
- Second, third, fourth, etc… arguments are what to print.
  - if omitted, prints what ever is in variable $_

Output examples

- Hello World program:

```
#!/usr/bin/local/perl
print “Hello World\n”;
```

- as this is perl, you can put string in paren’s, but you don’t need to (usually – because this is Perl).
- more examples:
  - `print “My name is $name\n”;`
  - `print “Hi ”, “what\’s ”, “yours?\n”;`
  - `print 5 + 3;`
  - `print ((4 * 4). “\n”);`

Input from keyboard

- read line operator: <>
  - aka “angle operator”, “diamond operator”
  - Encloses file handle to read from. Defaults to STDIN, which is what we want.
- `$input = <>;`
  - read one line from STDIN, and save in $input
    - (See variable section, later in this presentation);
- @input = <>;
  - read all lines from STDIN, and save as array in @input
    - Again, this makes more sense later on…

Chop & Chomp

- When reading in, carriage return (“\n”) is included.
- Usually don’t want that.
- chomp will take off the last character of a string, if it is a “\n”.
- chop takes off last character of a string, regardless of what it is.
  - Hence, chomp is “safer”.
- `chomp ($foo = <>);`
  - Very common method of reading in one string from command line.

Variables

- Three (basic) types of variables.
  - Scalar
  - Array
  - Hash
- There are others, but we’ll talk about them at a later time….

Scalars

- Scalar = “single value”
- In C/C++, many many different kinds of single values:
  - int, float, double, char, bool
- In Perl, none of these types need to be declared
- Scalar variable can hold all these types, and more.

Scalars

- All Scalar variables begin with a $
- next character is a letter or _
- remaining characters letters, numbers, or _
- Variable names can be between 1 and 251 characters in length
- Ex: $foo, $a, $zebra1, $F87dr_df3
- Wrong: $24da, $hi&bye, barfoo

Scalar Assignments

- Scalars hold any data type:
- $foo = 3;
- $d = 4.43;
- $temp = ‘Z’;
- $My_String = “Hello, I’m Paul.”

Arrays

- Concept is the same as in C/C++
  - Groups of other values
  - Groups of scalars, arrays, hashes
- much more dynamic than C/C++
  - no declaration of size, type
  - can hold any kinds of value, and multiple kinds of values
- All array variables start with the @ character
  - @array, @foo, @My_Array, @temp34

Array assignments

- @foo = (1, 2, 3, 4);
- @bar = (“hello”, “my”, “name”, “is”, “Paul”);
- @temp = (34, ‘z’, “Hi!”, 43.12);
- Arrays are 0-indexed, just as in C/C++
- $temp[1] = ‘z’;
  - NOTE: This is a *single value*, hence the $
- $bar[3] = “was”;
  - @bar now: (“hello”, “my”, “name”, “was”, “Paul”);

Array vs. Scalar

- $foo = 3;
- @foo = (43.3, 100, 83, 15.12, “Hi!”);
- $foo and @foo have *nothing in common*.
- In fact, $foo has nothing to do with $foo[3];
- “This may seem a bit weird, but that’s okay, because it *is* weird.”
  - Programming Perl, pg. 54

More about arrays

- special variable for each array:
  - @foo = (3, 25, 43, 31);
  - $#foo == 3. Last index of @foo.
  - $foo[$#foo] == 31;
- This can be used to dynamically alter the size of an array:
  - $#foo = 5; #creates two null values on the end of @foo
  - $#foo = 2; #destroys all but the first three elements of @foo
- "Slices" – part of an array (or hash)
  - @bar = @foo[1..3]; # @bar==(25, 43, 31)
  - @bar = @foo[0,1]; #@bar ==(3, 25)

Join/Split

- Built-in Perl functions
- Split – split a string into a list of values
  - $BigString = "Hello,_I_am_Paul";
  - @strings = split /_/, $BigString;
  - # @strings = ("Hello,", "I", "am", "Paul");
- Join – join a list/array of values together
  - $BigString = join ' ', @strings;
  - # $BigString == "Hello, I am Paul";

Hash

- Analogous to C++ hashtable.
- aka "Associative Array" – ie, array not indexed by numerical sequence.
- list of keys and values.
- All hash variables start with %

Hash example

- Want a list of short names for months:

```
%months = (
   “Jan” => “January”
   “Feb” => “February”
   “Mar => “March”
   …
);
```

- reference by *curly* brackets…
  - Avoid confusion with array notation
- $month{“Jan”} == “January”;