4
More Perl

Numbers

- Signed integers - size is machine dependent (ex: 4 bytes for 32 bit machine)
- Unsigned integers - Perl will convert signed integers that are too large to unsigned
- Double-precision floating-point numbers - usually 8 bytes, which is about 16 decimal places
- Numeric Strings - a numeric string is a character string that corresponds to the print representation of a number. “3” for example

Conversion Specifiers

- %d - signed int
- %u - unsigned int
- %f - float
- %e - exponential format (scientific notation)
- Can also qualify number of digits to show: (ex: %.2f to show 2 digits of a float, or %.3e to show 3 digits in exponential format)
- Full list of supported conversion specifiers - http://perldoc.perl.org/functions/sprintf.html

Strings

- Perl strings do not have to end with the null character like a C string does
- Single-quoted
- Double-quoted
- Version Strings - only used to denote the version of the Perl script
- Numeric Strings
- Character Escaping

Single-quoted Strings

- No special meanings given to any particular characters
- Except. . .there are 2 exceptions to this rule:
  - The apostrophe - ‘I\’m sorry Dave, that\’s incorrect’ => I’m sorry Dave, that’s incorrect
  - The backslash - ‘D:\\Perl Scripts\\Lab2.pl’ => D:\Perl Scripts\Lab2.pl
- Basically escape sequences or escape characters don’t work in single-quoted strings unless they escape an apostrophe or a backslash, i.e. \’ or \\

Double-quoted Strings

- Single quotes are ordinary characters inside a double-quoted string
- Support for all kinds of escape characters - Ex: \n, \t, \U, \E, \x*, etc
- Support variable interpolation - When a scalar value (like a number) is present in a double-quoted string, the print representation of the value of the variable is substituted into the string.

Example of string interpolation:

$coconuts = 200;

$str = “It would take about $coconuts coconuts to make me sick”;

Numeric Strings

- It’s really a kind of number, not a string (even though it looks like a string)
- A string all of whose characters are digits. Ex: “1234”
- Perl is perfectly happy performing numeric operations using numeric strings or a combination of numbers and numeric strings, until the operations don’t make any sense. Ex: $var = “42apple” + 17;

String Operations

- Concatenation - the “dot” operator - Ex: “Charles” . “ton” => Charleston
- length()
- substr()
- Index()
- print()
- uc()
- lc()
- Can do even more using regular expressions

More about arrays

- Arrays and Hashes are Perl’s built-in containers
- Arrays can consist of different kinds of things
- List literals - Ex: (41, 72, “eggs”, “milk”, 17, -8, 4)
- Lists are simply ordered sequences of scalar values, and an array is a variable whose value is a list
- Uninitialized arrays are set to the empty list ()
- Arrays can contain other arrays as elements
- qw - the quote word operator allows assigning values to an array using a space-delimited string. Ex: @friends = qw/ dale tucker /;

More about arrays

- Perl allows negative array indexing
- The size of an array @arr (the number of elements it contains) can always be retrieved by saying $#arr + 1 because the scalar variable $#arr is the index of the last element
- shift operator - removes and returns the first element, like pop on a stack
- unshift operator - like push on a stack, returns the array length after
- Array slicing and splicing
- Perl makes working with arrays very easy :)

Displaying arrays

- String interpolation has a special role in displaying arrays when it is evaluated in list context. Ex: print "@arr\n";
- But notice how the . concatenation operator supplies scalar context.
  Ex: print $#arr . "\n";
- Uninitialized arrays print out as empty strings
- Foreach uses a special variable $_ to hold the current array element.
- The range operator can be used to return a list from an array slice.
  Ex: 0..$#arr returns the entire array named arr.

Displaying arrays

- It is an error to access an uninitialized array element, unless it is done as the argument to the defined() function.
- The sort() function defaults to sorting lexicographically (dictionary sort).
- The spaceship operator < = > is used for custom sort functions for numbers.
- The cmp operator helps with defining custom sort functions for strings.
- The special system-defined variables $a and $b are provided as aliases for the current elements being compared in the custom sort function.
- The custom sort function should never modify the values of $a or $b.

More about hashes

- Hash keys must be strings or barewords, the values can be any scalar value
- Perl has a couple of different ways to assign key-value pairs to a hash
  - Big arrow operator => using this method you don’t have to place quotes around key names
  - Using a list of words - this method assigned keys and values as they appear in the list
- Individual hash values are accessed and assigned using the following syntax: $hash_name{ key_name }
- New key-value pairs can be inserted simply by assignment
- The delete operator removes and key-value pair if it exists in the hash - Ex: delete $cookies{ chocolate_chip }

More about hashes

- Iterating over a hash using the each operator
- Fetching all the keys and then using the foreach structure
- The exists() function can be used to check if a key exists
- The keys function can return all the keys in list context and the total number of keys in scalar context
- Likewise, the values function can return all the values in list context and the total number of values in scalar context

Displaying hashes

- Hash-slice - a specific set of key-value pairs.
- Hashes can be displayed in many different ways - keys only, values only, keys and values, etc
- % does not possess any special meaning inside a double-quoted string, like $ and @ do. Therefore, you cannot print a hash just by saying print “%hash\n”;
- One interesting use of scalar context is when the . operator (concatenation) is supplied to a hash it prints out the number of occupied buckets over the number of allocated buckets in the hash. Ex: print %hash . "\n";   prints 3/8

Terminal I/O

- Perl idioms are heavily influenced by Unix. Unix and Linux are based on the file abstraction. Everything is basically a file in Unix. So in Perl the standard streams STDIN, STDOUT, and STDERR are called *filehandles*.
- It is also helpful to remember that these streams can be redirected or “attached” to other things such as a file, a device, or another stream.
- The default output stream for the print function is STDOUT, so we don’t have to explicitly say print STDOUT “hello world”; every time.
- We can demo redirecting the output of STDERR to a file by running StdErr.pl as “perl StdErr.pl 2> log.txt”

File I/O

- The open() function attaches a stream (or filehandle) to a disk file, a device, another process, etc
- If open fails it returns the null object undef. We can say “or die” to handle this case and provide the user an appropriate error message, which will be redirected to the STDERR stream.
- The special scalar variable $! At the end of the string argument to die will be set to a system-supplied error message that may provide more information.

File reading and writing modes

- Use < to read input from a file, and > to output data to a file.
- Use >> to append data to an existing file without overwriting data already in it.
- Use +< to open a file for both reading and writing an existing file.
- Use +> to create a new file for both reading and writing.
- Use +>> to open either a new file if it does not exist or an existing file of the same name.
- Use the seek() function to set the position of the file pointer. This position is where the next read or write operation will take place in the file (in bytes).

Scope in Perl

- Global variables
  - System-supplied truly global variables - $_ for example
  - Visible everywhere after its declaration, if called with its package-qualified name
- Lexical variables
  - Defined using the my declaration. Ex: my $pizza = 14;
  - Lexical variables have lexical scope - they are only visible the point of their declaration and the end of the lexical scope, also known as static scope.
- Lexical scope usually means the area defined by a { and a }, but can also be defined as the string argument to the eval operator
- Lexical scopes can be nested

Reading Assignment

- Read the sections of the document at https://qntm.org/files/perl/perl.html starting with the section titled “References and nested data structures” and going through the sections titled “Modules and packages”