```
 1:
 2: $Id: Perl-notes,v 1.4 2012-01-06 19:09:23-08 - - $
 3:
 4: Source ideas: http://stuff.mit.edu/iap/perl/slides/slides.html
 5:
 6: Perl summary:
 7: - glue of the internet
 8: - high level language
 9: - builtin dynamic arrays and hashes
10: - scripting language
11: - no compilation needed
12: - powerful for text processing
13: - Unix system interfaces builtin
14: - other languages similar: Python, Ruby
15: - created by Larry Wall
16:
17: Examples:
18: % perl -e 'print "hello world\n"'
19:
20: % cat hello.perl
21: #!/usr/bin/perl
22: print "Hello, world!\n"
23:
24: - remember to chmod +x all perl scripts
25: - the first line must be the hashbang
26:
27: Example sieve of Eratosthenes:
28: - INEFFICIENT:  runs in time O(n^2)
29:
30: $maximum = 1000;
31: @numbers = (2..$maximum);
32: while ($prime = shift @nubmers) {
33:    print $prime\n";
34:    @numbers = grep {$_ % $prime != 0} @numbers;
35: }
36:
37: ----------------------------------------------------------------
```

```
38:
39:
40: Running perl:
41: create a file foo.perl using any text editor.
42: Make sure the first line is a hashbang:
43:
44: #!/usr/bin/perl
45:
46: - no white space
47: chmod +x foo.perl
48: run it with:  foo.perl
49: - just give its name on the command line.
50: - some programs will not have a suffix
51:
52: If the program is non-trivial:
53: make sure to
54:
55: use strict;
56: - requires declaration of variables and checks on other things
57: use warnings;
58: - prints extra warnings, specially for uninitialized variabls.
59:
60: What is true?
61: - anything that is not false.
62: What is false?
63: - undef, 0, ""
64:
65: # Comments start with a # character
66:
67: Data types:
68: scalars - singe values: strings, numbers, references.
69:         - types are dynamic and can change over time
70: arrays  - multiple values indexed by integers starting from 0
71:         - @a in scalar context is like a.length in Java.
72:         - $a[$i] is an index element
73: hashes  - %h is a hash, $h{$k} is an element of a hash
74:         - built into perl, so symbol table applications are easy
75: data structures - created by scalars, arrays, hashes of references
76:
77: ----------------------------------------------------------------
```

```
 78:
 79:
 80: Sigils: $, @, %
 81: Identifiers are case-sensitive
 82: Also special names:  $0, @ARGV, $_, $!, $., $&
 83:
 84: Scalars:
 85:    $sigil
 86:    - may contain numbers (integer or double)
 87:    - strings
 88:    - references
 89:
 90: Strings:
 91:    - bounded by 'abc', "abc", or `abc`
 92:    - '$abc\n' is the literal strings itself
 93:    - "$abc\n" is the value of the variable $abc followed by nl char
 94:    - `time` is the output of running the command time by the shell
 95:
 96: Local variables declared by my:
 97:    sub foo () {
 98:       my $a = 6; # lexically scoped as in Java
 99:    }
100:
101: $a = 6; $a = "abc"; ... type may change over time
102:
103: ---------------------------------------------------------------------
```

```
104:
105:
106: ARRAYS:
107:    @a for a whole array
108:    $a[$i] for an element of an array
109:    size may change over time
110:    @a[$i..$j] is a slice of an array
111:    $a[-1] is the last element , $a[-2] second last
112:    $a[0] is the first element
113:
114: for (my $i = 0; $i < @a: ++$i) { ... }
115: - when @a in scalar context, like Java a.length
116: for my $i (@a) { ... }
117: - like Java for (i: a) {...}
118:
119: @a = @b is whole array assignment (copying), not reference assignment
120:
121: $x = pop @a; removes last element assigns to $x
122: push @a, $x; appends $x to end of array
123: - stack is trivial
124:
125: $x = shift @a; removes 0 element assigns to $x
126: unshift @a, $x; inserts new front element and shoves down the rest
127: - queue is push and shift
128:
129: Also has a splice operation
130:
131: With subscript, sigil changes to $ if a scalar is produced:
132: $a[$i] is a scalar, @a[1,10,20] is an array with element selected
133: $#a is usually @a - 1;
134:
135: @a = (1, 3, 5, 7, 9); - initializes an array
136: @a = (1..10, 101..110); - array has 20 elements in it
137:
138: $a[10000] if no element returns undef
139:
140: ----------------------------------------------------------------
```

```
141:
142:
143: HASHES:
144:    store any number of key/value pairs
145:    %sigil for hash
146:    $h{$k} for an element
147:    @h{$k,$j} for an array of selected values
148:    @h{@a} returns an array of values indexed by elements @a
149:
150: %h = ("foo"=> 3, "bar"=> 5);
151: - initialized by an array with alternating key=>value pairs.
152: for my $key (sort keys %h) { ... $h{$key}... }
153:
154: $h{$k} returns undef if not found
155: - undef is sort of like null in Java
156:
157: delete $h{$k}; removes a key/value pair
158:
159: CONTEXT:
160: - no difference between '1', "1", and 1
161: - auto stringifies numbers when needed, auto numifies strings
162: - context determines value
163: - context may be scalar or list.
164: - list==array, so linked lists are never needed in Perl
165: - $a = "33foo" + 6; sets $a to 39
166:
167: String context:
168: - dot (.) is concatenation
169: - $a = 33 . 44; concatenation => 3344
170: - "" interpolates \escapes, $i and @a values
171: - @a = (1,2,3); $s="@a"; $s = '1 2 3';
172:
173: Boolean context:
174: - undef, 0, "" are false; everything else is true
175:
176: ----------------------------------------------------------------
```

```
177:
178:
179: OPERATOS:
180:    numeric, string, quoting, boolean, list
181:
182: man perlop:
183:             left        terms and list operators (leftward)
184:             left        ->
185:             nonassoc    ++ --
186:             right       **
187:             right       ! ~ \ and unary + and -
188:             left        =~ !~
189:             left        * / % x
190:             left        + - .
191:             left        << >>
192:             nonassoc    named unary operators
193:             nonassoc    < > <= >= lt gt le ge
194:             nonassoc    == != <=> eq ne cmp
195:             left        &
196:             left        | ^
197:             left        &&
198:             left        ||
199:             nonassoc    ..  ...
200:             right       ?:
201:             right       = += -= *= etc.
202:             left        , =>
203:             nonassoc    list operators (rightward)
204:             right       not
205:             left        and
206:             left        or xor
207:
208: . concatenation
209: x repeititon:  $x = "-" x 10 means $x = "----------"
210: == != < <= > >= numeric operators
211: eq ne lt le gt ge string operators like (strcmp)
212: cmp is string.compareTo
213: <=> is numeric.compareTo
214: So:  19 > 9, but "19" lt 9
215:
216: ----------------------------------------------------------------
```

```
217:
218:
219: Quoting operators
220: "" causes interpolation of $x scalars, @a arrays, \n, etc. escapes
221: '' are for literal strings only, no interp
222: `` interpolates passes string to shell and captures results.
223: $i = 6 @a = (1,2,3);
224: $s = '$i @a\n'; literal string
225: $s = "$i @a\n"; string is '6 1 2 3' followed by nl char
226: $s = `ls`; $s has all files in current dir.
227:
228: List operators
229: push, pop, shift, unshift - see above
230: sort @a - sorts array lexicographically
231: sort {$a <=> $b} @a - sorts array numerically
232: - can use any block as first argument to sort
233: reverse @a - reverses the list
234: split /,/, $s - splits a string into an array with , as delim
235: join "," @a - makes a string from elements of array with , interpolated
236: grep {boolexpr} @a - returns a subarray for which boolexpr is true
237:    $_ special variable "it".
238: @b = grep {$_ > 0} @a; sets @b to all positive elements of @a
239:
240: map {expr} @a - maps each expr onto elements of araray
241: @b = map {$_ * 2} @a; sets b - double of @a elements
242: map {$set{$_} = 1} @array - creates a hash with keys from the array
243: - all values are 1.
244:
245: ----------------------------------------------------------------
```

```
246:
247:
248: Flow of control:
249: if (expr) {stmts...}
250: unless (expr) {stmts...}
251: if (expr) {stmts...} elsif (e) {stmts...} elsif (e) {stmts...} else{s}
252: - the {} on statements are required.
253: $a = "foo" if expr;
254: - can use if or unless as a suffix.
255:
256: while(e){stmt...}
257: simple while e;
258: for (init; test; step) {stmt...}
259: for var (list) {stmt...}
260: simple for $a;
261: print "$_\n" for 1..10; - prints out numbers 1..10;
262: for===foreach
263: && || !? and or - all short circuit operators.
264:
265: do {stmt} until expr;
266: do {stmt} while expr;
267:
268: next - like continue
269: last - like break
270:
271: ---------------------------------------------------------------------
```

```
272:
273:
274: Subroutines
275: sub add {
276:    my ($a, $b) = @_;
277:    return $a + $b;
278: }
279: - value of sub is value of last statement or return value
280: - @_ is array of arguments
281:
282: $a = add (3, 4);
283: $a = add 3, 4;
284: parents are not always necessary
285:
286: sub update {
287:    my ($a) = @_;
288:    $$a = $$a + 1; # like C: *a = *a + 1
289: }
290: update (\$a); like C update (&a);
291:
292: my $a = @a instead would set $a to the length of @_
293: - parens needed here
294:
295: ----------------------------------------------------------------
```

```
296:
297:
298: References:
299: used to make compilicated data strutores.
300: Example trees:
301:
302: $t = {LEAF=> 'a'};
303: - hash refernce
304: $u = {LEAF=> 'b');
305: $v = {OPER=> '+', LEFT=>$t, RIGHT=>$u};
306: {} makes a hash ref, [] makes an array ref.
307: $p = \@a; makes $p point at an array
308: $p->[$i] dereferences that element.
309: $q = \%h; address of a hash
310: $q->{$k} selects from the hash
311: No structs, so use hash references instead.
312:
313: $pa = [1, 2, 3]; reference to an array
314: $ph = {3, 4, 5, 6}; reference to a hash
315: A reference is always a scalar
316:
317: ----------------------------------------------------------------
```

```
318:
319:
320: FILES:
321:
322: Access to files uses open and close
323: open my $file, "<$filename" or warn "$0: $filename: $!\n" and next;
324: - opens the file for reading and autovivifies the variable $file
325: - $! is like strerror(errno)
326:
327: open my $file, ">$filename" opens it for output
328: close $file closes the file.
329:
330: Standard files: STDIN, STDOUT, STDERR
331: print $a, "\n" -- uses STDOUT
332: print STDERR "whatever"; is like fprintf (NOTE: no comma
333: print $file "foo"; prints ot $file (no comma)
334:
335: $a = <$file> reads a line from $file;
336: - undef at EOF
337:
338: while ($line = <>) {......}
339: reads lines from all of the files specifiedin @ARGV if any,
340: or STDIN if not.
341:
342: while (<>) {print}
343:
344: same as
345: for my $filename (@ARGV) {
346:    open my $file, "<$filename" or warn "$0: $filename: $!\n" and next;
347:    while (defined (my $line = <$file>)) {
348:       chomp $line;
349:       print $line, "\n";
350:    }
351:    close $file;
352: }
353:
354: ----------------------------------------------------------------
```

```
355:
356:
357: PIPES:
358: open my $lsla, "ls -la |" or die "$0: ls -la: $!\n";
359: -- opens a pipe output from the ls -la commend.
360:
361: open my $fmt, "| fmt" or die "$0: fmt: $!\n";
362: -- prepares FH $fmt for writing to the fmt command.
363:
364: $a = `ls`; implicit pipe, stdout captured in a string
365: @a = `ls`; implicit pipe, each stdout line in a separate element.
366:
367: File check:
368: if (-f $filename) - check if f is a plain file
369: if (-d $dir) - checks for dir.
370: -- see more tests in man perlop (perlfunc?)
371:
372: ----------------------------------------------------------------
```

```
373:
374:
375: Regular expressions
376: Pattern matching based on Chompsky's type 3 languages.
377:
378: If ($a =~ m/foo/) - if $a contains the string foo
379: if ($a !~ m/foo/) - if $a does NOT match foo
380: $a =~ s/foo/bar/ - change the first occurrence of foo to bar
381: $a =~ s/foo/bar/g - change all occurrences of foo to bar
382:
383: $0 =~ s|.*/||; delete all chars before the last /
384:     . match any character
385:     * zero or more
386:     ||| substituted delimiter chars
387:     delim char may not appear in pattern
388: $0 =~ s/.*\///; - same as above
389:
390: Regexes have chars and metachars.  \ swaps the category.
391: \d digits [0-9], \D non-digits [^0-9]
392: \w word chars [a-zA-Z0-9_], \W non-word chars
393: \s white space [ \t\n], \S non-whitespace
394: .  any char same as [^\n]
395: x*  zero or more x, greedy
396: x+  one or more x, greedy
397: x?  optional x, same as (x|)
398: x*? zero or more x, non-greedy
399: x+? one or more x, non-greedy
400: x|y indicates alternation
401: [xyz] means (x|y|z)
402: [a-z] a or z or anything in between (lexicographic order)
403: [^abc] any character not a or b or c
404: ^ beginning of string
405: $ end of string
406: x{3,10} 3 to 10 x
407: x{4} exactly 4 x
408: \*, \+, \? matches the character itself
409:
410: ------------------------------------------------------------------
```

```
411:
412:
413: Substitutions and capture.
414: $a =~ s/(.*), (.*), (.*)/$3, $2, $1/;
415: changes "abc, def, ghi" to "ghi, def, abc"
416:
417: if ($date =~ m/^(\d+)\/(\d+)\/(\d+)$/) {
418:    my ($month, $day, $year) = ($1, $2, $3);
419:    my $wholedate = $&;
420: }
421:
422: $a =~ m/$x/i; matches value of $i case insensitive
423: $a =~ s/abc/def/g; global substitution
424:
425:
```