Topic 3

Lists and arrays

Learning Perl 2nd edition
chapter 3, pages 48-57
Programming Perl 3rd edition
pages 69-76
Programming Perl 2nd edition
pages 47-49
`perldata` manpage

1

Last time

Covered in Topic 2

- Scalar values
  - numbers
  - strings
- Scalar variables
- Scalar operators
- Console input/output
  - printing to the screen
  - reading from the keyboard
- Interpolating into strings

2

To be covered today

- Lists
- Arrays
  - variables that contain lists
- List and array functions
  - sorting lists
  - adding and removing array elements
- Context
  - scalar versus list

3

Lists

Sequences of scalars

- A ordered sequence of scalars
  - each element may be any scalar expression, including variables or literals
- List literals enclosed in parentheses
  - `(-5.3, 42, "porcupine", $a+10)`
    - last element uses value of `$a` at the time the list literal is used
- Each element has a position (index)
  - first element is at index 0

Llama2 pages 48-49; *Camel3* pages 8-10, 72-75
Camel2 pages 6-7, 47; `perldata` manpage

4

Arrays

Relationship to lists

- An array is a Perl variable containing a list
- All Perl arrays begin with the character `@`
  - `@names`
  - `@temperatures`
  - `@_`
- Arrays are assigned (copied) with `=`
  - `@days = ("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");`
  - `@copy_of_days = @days;`

Llama2 pages 48-49; *Camel3* pages 8-10, 51;
Camel2 pages 6-7, 47; `perldata` manpage

5

Arrays

Things to note

- Lists cannot hold arrays inside them
  - arrays named inside lists are “flattened”
  - `@weekdays = ("Mon", "Tue", "Wed", "Thu", "Fri");`
  - `@days = ("Sun", @weekdays, "Sat");`
  - `@days` now contains seven elements `("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")`
  - nested data structures require references (Topic 11)
- Array names and scalar names are separate
  - `$days` and `@days` are unrelated

Llama2 page 51; *Camel3* pages 52, 73; *Camel2* page 47

6

Array elements

Accessing a single element of `@array`

- `$array[`index`]`
  - `$days[1]  # equal to "Mon"`
- Index may be any scalar (integer) expression
  - If index is a scalar variable, still needs $ sign
    - `$array[$i]`
  - first element has index 0
  - last element has index `$#array`
    - `$#array` is always (size of `@array` – 1)
  - negative indices count from right hand end of array
    - `$array[-1]` is the same element as `$array[$#array]`

Llama2 page 52; *Camel3* pages 8-9; *Camel2* page 7

Array elements

Accessing a single element of `@array`

- `$array[`index`]`
- One array element is of scalar type
  - thus `$` to indicate one element of `array`
  - `$days[1]` has nothing to do with `$days`
  - `@days[1]` is an array slice, which is probably not what you want
    - array slices are used to select several array elements at once
    - `@days[1, 2, 3, 4, 5]` (equal to list `("Mon", "Tue", "Wed", "Thu", "Fri")`)
    - value of an array slice is a list
- If array index is out of bounds, value is `undef`
  - `$days[7] # equal to undef`

Interpolating arrays

Putting an array or array element into a string

- Like scalars, arrays can be interpolated into double-quoted strings
  - `$alldays = "@days";`
  - `$alldays` is equal to `"Sun Mon Tue Wed Thu Fri Sat"`
  - Each element is separated by one space
- Array elements can also be interpolated
  - `print "I don't like $days[1]\n";`
- As usual, use braces to disambiguate
  - `$days = "schooldays";`
  - `print "I don't like ${days}[1]\n";`
  - prints “I don't like schooldays[1]”

Llama2 page 56-57; *Camel3* pages 65-66; *Camel2* page 43

Operating on lists

Some list functions

- print
  - prints each element of list in turn
  - `print "Hello ", $name, "\n";`
  - `print @lines;`
- sort
  - returns an alphabetically sorted list
  - `@sorted_day_names = sort @days;`
  - `print sort (3, 2, 1);  # prints "123"`
- reverse
  - returns a list with the elements in reverse order
  - `@flip = reverse @flip;`

Llama2 pages 54-55; *Camel3* chapter 29
Camel2 chapter 3; `perlfunc` manpage

Operating on arrays

Some array functions

- `push, pop`
  - adds elements to or removes an element from the right hand side of an array
  - `@array = (1, 2, 3, 4);`
  - `push @array, 5;      # Now 1, 2, 3, 4, 5`
  - `$five = pop @array; # Now 1, 2, 3, 4`
  - `push` can add several items at once
- `unshift, shift`
  - adds elements to or removes an element from the left hand side of an array
  - `unshift` has same syntax as `push`; `shift` has the same syntax as `pop`

Llama2 page 54; *Camel3* chapter 29;
Camel2 chapter 3; `perlfunc` manpage

Example

Printing a reverse-sorted list

```
# Original list of reindeer.
# An alternative syntax:
# @reindeer = qw(Dasher Dancer Prancer
#   Vixen Comet Cupid Donner Blitzen);
@reindeer = ('Dasher', 'Dancer', 'Prancer',
'Vixen', 'Comet', 'Cupid', 'Donner', 'Blitzen');

# Rudolph, with your nose so bright ...
unshift @reindeer, "Rudolph";  # Add to front

# Sort the list and reverse it.
@reindeer = reverse sort @reindeer;

# Print them out.  Prints:
# Vixen Rudolph Prancer Donner Dasher
# Dancer Cupid Comet Blitzen
print "@reindeer\n";
```

Context

An ambiguity

- In C (and Perl), comma operator can be used to perform two operations
  - `for (i=1, j=1; a[i] == a[j]; i++, j++)`
  - value of `a,b` is value of `b`
  - parentheses can be used for grouping
  - so value of `(a,b)` is value of `b`
- So is `($apples, $oranges, $pears)` a three-element list or the single value `$pears`?

Llama2 page 55; *Camel3* pages 69-72
Camel2 pages 45-46; `perldata` manpage

13

Context

The solution

- Depends on context of code
- Context means what is expected at a certain point in the program (scalar or list)
- List context: if list expected
  - `@a = ($apples, $oranges, $pears);`
  - treated as a list because assigning to array
  - `@a` receives a three-element list
- Scalar context: if scalar expected
  - `$a = ($apples, $oranges, $pears);`
  - treated as a scalar because assigning to scalar
  - `$a` receives value of `$pears`

14

Context

Expectations of operators and functions

- Some operators and functions need a scalar
  - `length`, `+`, `rand`, `.`
  - force scalar context on their arguments
- Some operators and functions need a list
  - `print`, `push`, `sort`
  - force list context on their arguments
- Some operators and functions don't care
  - `reverse`, `<STDIN>`, `chomp`
  - use whatever context they are given
  - may produce different results depending on context

15

Context

Using a scalar in list context

- What if a scalar is used where a list is expected?
  - `@array = $kiwifruit;`
  - assigning to array, so list context
  - scalar is promoted to a one-element array
  - same as `@array = ($kiwifruit);`
  - `print "Hello world";`

16

Context

Using an array in scalar context

- What if an array is used when a scalar is expected?
  - `$scalar = @fruitsalad;`
  - assigning to scalar, so scalar context
  - array evaluated in scalar context is converted to size of array
  - `$scalar` receives number of elements in `@fruitsalad`
  - scalar context can be enforced with `scalar` function
    - `print scalar @days;    # prints 7`
  - no general rule for converting any list function result into scalar
    - some functions have their own rules

17

Using context

Reversing all lines input

```
# Read in all lines.
# <STDIN> in list context reads until EOF,
# one line per list element.
@lines = <STDIN>;

# Reverse @lines array.
@backwards = reverse @lines;

# Print @backwards array.
print @backwards;

# This entire program could be written
# as one line:
# print reverse <STDIN>;
```

18

Covered today

- Lists
  - sequences of scalars
- Arrays
  - variables that contain lists
  - always start with `@` character
- List and array functions
  - `sort`, `reverse`
  - `push`, `pop`, `shift`, `unshift`
- Context
  - scalar context when scalar expected
  - list context when list expected

Going further

More things related to today's topic

- References
  - nested data structures
  - Topic 11
- `map` and `grep`
  - useful list and array functions
  - *Camel3* pages 740-741, 730; *Camel2* pages 186-187, 178-179

Next time

to be covered in Topic 4

- Control structures
  - `while`, `if`, `foreach`, etc.
- True and false values
- Statement modifiers
  - An alternative way to do some control structures
- Perl defaults
  - `$_`, `<>`

Reading:
Learning Perl 2nd edition chapter 4, pages 72-73
Programming Perl 3rd edition pages 111-127, 658-659, 80-83
Programming Perl 2nd edition pages 95-105, 131, 53-55
`perlsyn`, `perlvar` manpages