

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

CSE2395/CSE3395

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)

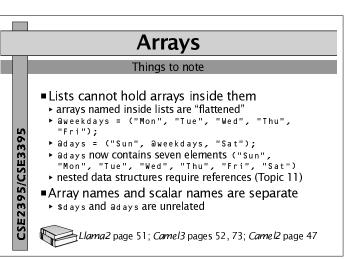
• first element is at index 0

\*\*Llama2\* pages 48-49; \*Came13\* pages 8-10, 72-75
\*\*Came12\* pages 6-7, 47; \*perldata\* manpage

4

## Relationship to lists An array is a Perl variable containing a list All Perl arrays begin with the character a anames atemperatures all Arrays are assigned (copied) with = adays = ("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"); acopy\_of\_days = adays; Llama2 pages 48-49: Camel3 pages 8-10, 51; Camel2 pages 6-7, 47; perldata manpage

3



## **Array elements**

### Accessing a single element of @array

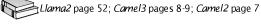
- sarray[index]
  - ▶ \$days[1] # equal to "Mon"
- Index may be any scalar (integer) expression
  - ▶ If index is a scalar variable, still needs \$ sign
    - \$array[\$i]

CSE2395/CSE3395

CSE2395/CSE3395

CSE2395/CSE3395

- ► 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]



7

## **Array elements**

### Accessing a single element of parray

■ \$array[index]

CSE2395/CSE3395

- One array element is of scalar type
- → thus \$ to indicate one element of array
- ▶ \$days[1] has nothing to do with \$days
- ► adays[1] is an array slice, which is probably not what you want
- array slices are used to select several array elements at once
- adays[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

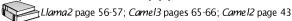
8

**Operating on lists** 

## **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]"



9

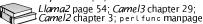
10

## **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

11



## **Example**

### Printing a reverse-sorted list

# Original list of reindeer.
# An alternative syntax:
# areindeer = qw(Dasher Dancer Prancer

# Vixen Comet Cupid Donner Blitzen);
areindeer = ('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";

12

CSE2395/CSE3395

## An ambiguity In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operations In C (and Perl), comma operator can be used to perform two operators In C (and Perl), comma operator can be used to perform two operators In C (and Perl), comma operator can be used to perform two operators In C (and Perl), comma operator

The solution

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

13

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, <SIDIN>, chomp

use whatever context they are given

may produce different results depending on context

Using a scalar in list context

What if a scalar is used where a list is expected?

Description array = \$kiwifruit;
Description array, so list context
Description array = (\$kiwifruit);
Description array = (\$kiw

15 16

# 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

```
Reversing all lines input

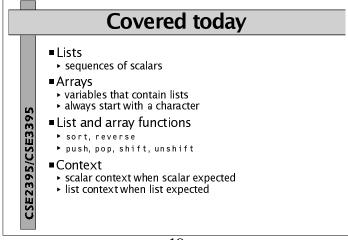
# Read in all lines.
# <STDIN> in list context reads until EOF,
# one line per list element.
alines = <STDIN>;

# Reverse alines array.
abackwards = reverse alines;

# Print abackwards array.
print abackwards;

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

17 18



Going further

More things related to today's topic

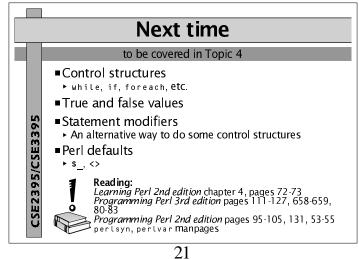
References

• nested data structures
• Topic 11

■ map and grep

• useful list and array functions
• Camel3 pages 740-741, 730; Camel2 pages 186-187, 178-179

19 20



CSE2395/CSE3395 lecture notes copyright © 2000-2001 Deborah Pickett.
Reproduction of this presentation for nonprofit study use is permitted. All other reproduction, including for other educational courses, must be authorized in writing by the author.

CSE2395/CSE3395