

CS306: Introduction to Perl

Lists and Arrays

Lists and Arrays

U. of Alabama at Birmingham
Dept. of Computer & Information Sciences

Introduction
Accessing Arrays
Context

Slide 1

Slide 2

Lists

- A *list* is an ordered collection of scalars. We call each individual scalar an *element* of the list.
- The elements of a list are *indexed* starting at 0.
- Each element is independent scalar which means that they can contain anything a scalar can
- The elements don't have to hold the same kind of data, one could be a string, another a number, another undef, etc...

Slide 3

Creating List Literals

- Use the () notation
- ("fran", "ying", "tony")
- (1, 5, 3, 4)
- ("fran", 2, undef, "seven", 4.5)
- () # The empty list
- (1, 4, 2,)
 - Trailing comma ignored. This enables a common syntax usage we'll see later.

Slide 4

Other Ways To Create Lists

- The `..` *range* operator
 - `(1..10)` # Creates a 10 element list
 - `('A'..'Z')` # The alphabet in capital letters
 - `('aa'..'zz')` # Permutations (“aa”, “ab” ... “zy”, “zz”)
- The `qw` shortcut - “quoted words”
 - `qw /george jane judy elroy/`
 - Other delimiters like `!!` `{}` `[]` `<>` `()` also work

Slide 5

Arrays

- Lists aren't very useful unless we can store the data somewhere, so we have *arrays*.
- Arrays are variables which store lists. The array is the container; the list is the data. All arrays are lists, but not all lists are arrays.
- An array with 0 elements is said to have a value of the empty list, or `()`. Note this is not `undef`.

Slide 6

Creating Arrays

- Perl uses the `@` symbol to indicate an array
- `@names`
- Note that this is a completely different variable than `$names`, and they can co-exist, unaware of each other
- Same naming rules as for scalars (alphanumerics, underscores, can't start with a digit, etc...)

Slide 7

Assigning Lists to Arrays

- `@names = ("fran", "tony", "ying");`
- `@colors = qw[red yellow orange purple];`
- `@morenames = ("john", @names, "amy");`
 - Note that `@morenames` is a 5 element array. Array elements can only be scalars, not other arrays. `@names` is expanded before the assignment happens.

Slide 8

Accessing Elements

- Access the elements with [index] notation
- `$names[0]`, `$names[1]`, etc...
 - Note it's `@names`, but `$names[0]`. Remember, each element is a scalar, so we use `$` when referring to just one element and `@` for the whole array.
 - Even trickier, `$names[0]` is a completely separate variable than `$names`, and these too can co-exist just fine (but it's usually a bad code design to do this)

Slide 9

Accessing Elements 2

- Assign to a scalar:
 - `$name = $names[0];`
- Copy arrays
 - `@names = @people;`
- Assign to a bunch of scalars at once
 - `($make, $model, $color) = @carstats;`
 - `($one, $two) = ($two, $one);`
 - Swap variables, no `$temp` needed!

Slide 10

Weird Array Magic To Know

(...but probably not use too much)

- Grab the last element in two weird ways
 - `$names[$#names]`
 - `$#names` is shorthand for “the index of the last element of the `@names` array”. I told you perl loves shortcuts.
 - `$names[-1]`
 - Count from the end of the array. `-1` is the last element. Given a 3-element array, `-2` and `-3` also work, but `-4` doesn't, there's no wrap-around
- You may see these, but I discourage their use. Better way to do this is coming soon...

Slide 11

Slices

- Slices grab some elements of a list and return a list of just those elements
 - `@somenames = @names[2..4];`
 - `($date, $time) = @array[5,8];`
 - `@oddfields = @array[1,3,5,7];`
 - `($teacher, $ta) = (“fran”, “tony”, “ying”)[0,2];`
 - `($first, $last) = (sort @nums)[0,-1]; # Cool hack!`

Slide 12

Functions For Arrays

- *pop()* and *push()* – these operate on the right side of an array (a stack or LIFO)
 - `$lastelement = pop @array; # @array is now shorter`
 - `push @array, $new_element_on_right;`
- *shift()* and *unshift()* – these operate on the left side of an array (a pipe or FIFO)
 - `$firstelement = shift @array;`
 - `unshift @array, $new_element_on_left;`

Slide 13

Functions for Lists

- *reverse()* – reverses the elements of a list
 - `@ascending = reverse @descending;`
- *sort()* – sorts the elements of a list
 - `@sorted = sort @names;`
 - sort is an extremely powerful/general/flexible operator – it can sort any kind of data. In its most simple form, it sorts via string comparison.
 - `sort (98,99,100)` produces `(100,98,99)` # careful!

Slide 14

Functions for Lists 2

- *join()* – join a list into a string using a separator
 - `@names = ("fran", "tony", "ying");`
 - `$string = join 'l', @names; # "franltonylying"`
 - `$string = join @names; # "fran tony ying"`
- *split()* – splits a string into a list on a separator
 - `$string = "franltonylying";`
 - `@names = split //, $string;`

Slide 15

Interpolating Into Strings

- An entire array
 - `@teams = qw / blazers tide tigers /;`
 - `print "gators @teams dawgz";`
 - # produces "gators blazers tide tigers dawgz"
- Just one element
 - `print "The $teams[0] made the NCAA Tournament last year."`

Slide 16

It's All In The Context

- Perl has something very important called *context*
- Context simply means “What is Perl expecting here?”
- `8 - ??? # Here Perl expects a scalar`
- `sort ??? # Here Perl expects a list`
- `???` could be the same exact thing in both instances, but lead to a different answer, because it was used in a different context. **IMPORTANT.**

Slide 17

Context Examples

- `@heros = qw/ bond luke indy lara/;`
- List context
 - `sort @heros # produces “bond indy lara luke”`
- Scalar context
 - `5 + @heros # produces 9`
 - Since you used an array in scalar context, Perl assumed you meant the length of the array

Slide 18

Context Examples 2

- Assignment
 - `@array = “foo”;` # creates one element list
 - `$size = @array;` # \$size gets the size of the array
- Some functions do different things depending on the context in which you use them
 - `@ascending = reverse @descending;`
 - `$name = reverse “fran”;` # produces “narf”
 - `$name = reverse (“fran”, “fabrizio”);` # “oizirbafnarf”

Slide 19

Context Examples 3

```
$result = ???  
@array = ???  
($name, $address) = ???  
($result) = ???  
$colors[3] = ???  
push @array, ???
```

Slide 20

A Special Context Case - <STDIN>

- <STDIN> in scalar context
 - `$line = <STDIN>;`
 - `chomp $line;`
- <STDIN> in list context
 - `@lines = <STDIN>;` # grabs all lines until EOF
 - `chomp @lines;` or `chomp(@lines = <STDIN>);`
 - Use Ctrl-D or Ctrl-Z (Windows) to send EOF from keyboard