

Arrays and Lists



Lists, Arrays, Operators

- List: an ordered set of scalars
- Array: a variable that holds a list
- Each element of an array is a scalar
- Empty array has zero (0) elements
- Largest only limited by virtual memory
 - Perl philosophy: no unnecessary limits

List Literals

```
(1, 2, 5, 10)           # list of integers  
( 'horse', "pig", "chicken" ) # of strings  
("x", 3, 5, 4, 'z')    # of chars  
($a, 17)               # variable and int
```

```
()                   # the empty list; has zero elements
```

```
(1..5)              # the list constructor function
```

```
('a'..'z')         # works for letters too!
```

Arrays (@) Hold Lists

- The '@' character indicates an array:
- (note: \$cities is not related to @cities!)

@cities

Initializing an Array

- Literal array assignment is one way.
- Commas separate values:

```
@things = ('apple', "cat", "dog", 'emu');
```

```
# or use the quote words (qw) function, which  
# uses spaces to delimit elements:
```

```
@things = qw(apple cat dog emu);
```

Array Assignment

```
# assign from literal list  
@numbers = (1, 2, 3);
```

```
# assign from other array  
@copy_numbers = @numbers;
```

```
# makes @one a 1-element list  
@one = (1);
```

```
# use qw function  
@one_two = qw(one two);
```

Array Assignment

```
@one_two = qw(one two);
```

```
@numbers = (4, 5, @one_two, 6);
```

```
# @numbers is now
```

```
# (4, 5, "one", "two", 6)
```

Array and List

- Array on either side
- List on either side

```
($x, $y, $z) = @array;  
@array = ($a, $b, $c);
```



Array Assignment

```
@nums = (2..5); # how many elements?
```

```
# add an element to end of @nums
```

```
@nums = (@nums, 6);
```

```
# add an element to beginning of @nums
```

```
@nums = (1, @nums); # now how many?
```

```
# NOTE: this is NOT the best way to
```

```
# add elements to the beginning/end of
```

```
# an array!
```

Array/List on the Left

```
# assign three variables
```

```
($four, $five, $six) = (4, 5, 6);
```

```
($x, $y) = ($y, $x); # Swap!
```

```
($x, @nums) = ($four, $five, $six);
```

```
# $x is now 4, @nums is (5, 6)
```

List Assignment

```
# $no_val gets undef here -- why?  
($first, @nums, $no_val) = (7, 8, 9, 10);  
  
# $y gets undef here -- why?  
($x, $y) = (9);
```

Group Challenge!

- Pair or Triple Up!
- Set \$x and \$y both to 5, in only 10 characters
- In 8 characters?
- In 14 characters, but only one equals sign
- Whitespace doesn't count against the limit.



Array Size

- An array taken in scalar context returns the number of elements:

```
@numbers = (1, 3, 5, 7, 9);
```

```
$size = scalar @numbers;
```

```
# same thing as above:
```

```
$size = @numbers;
```

Array element access

- Use this syntax to access individual array elements:

```
@ages = (7, 8, 9);
```

```
$ages[0]
```

```
$ages[1]
```



Array Element Access

```
@ages = (7, 8, 9);
```

```
$first = $ages[0]; # $first gets 7
```

```
$ages[0] = 5;      # @ages now is (5,8,9)
```

```
$second = $ages[1]; # $second gets 8
```

```
$ages[2]++;      # makes the array (5,8,10)
```

```
$ages[1] += 4;   # makes the array (5,12,10)
```

Group Challenge!

- Pair or Triple Up!
- Make a 3 element array in 11 characters
- Make a 100 element array in 12 characters
- Make a million element array in 10 characters
- Whitespace doesn't count.



undef

```
@numbers = (7,8,9);  
  
# $num will be undef. Why?  
$num      = $numbers[7];  
  
$numbers[3] = 'hi';  
$numbers[6] = 'world';  
  
# @numbers is now:  
# (7, 8, 9, 'hi', undef, undef, 'world')
```

Negative Subscripts

```
@numbers = (7,8,9);  
  
print "$numbers[-1]\n";  
# prints 9  
# (count -1 back from end of array)  
  
print $numbers[-2]; # prints 8  
  
$numbers[-1] = 12; # set last element
```

Group Challenge!

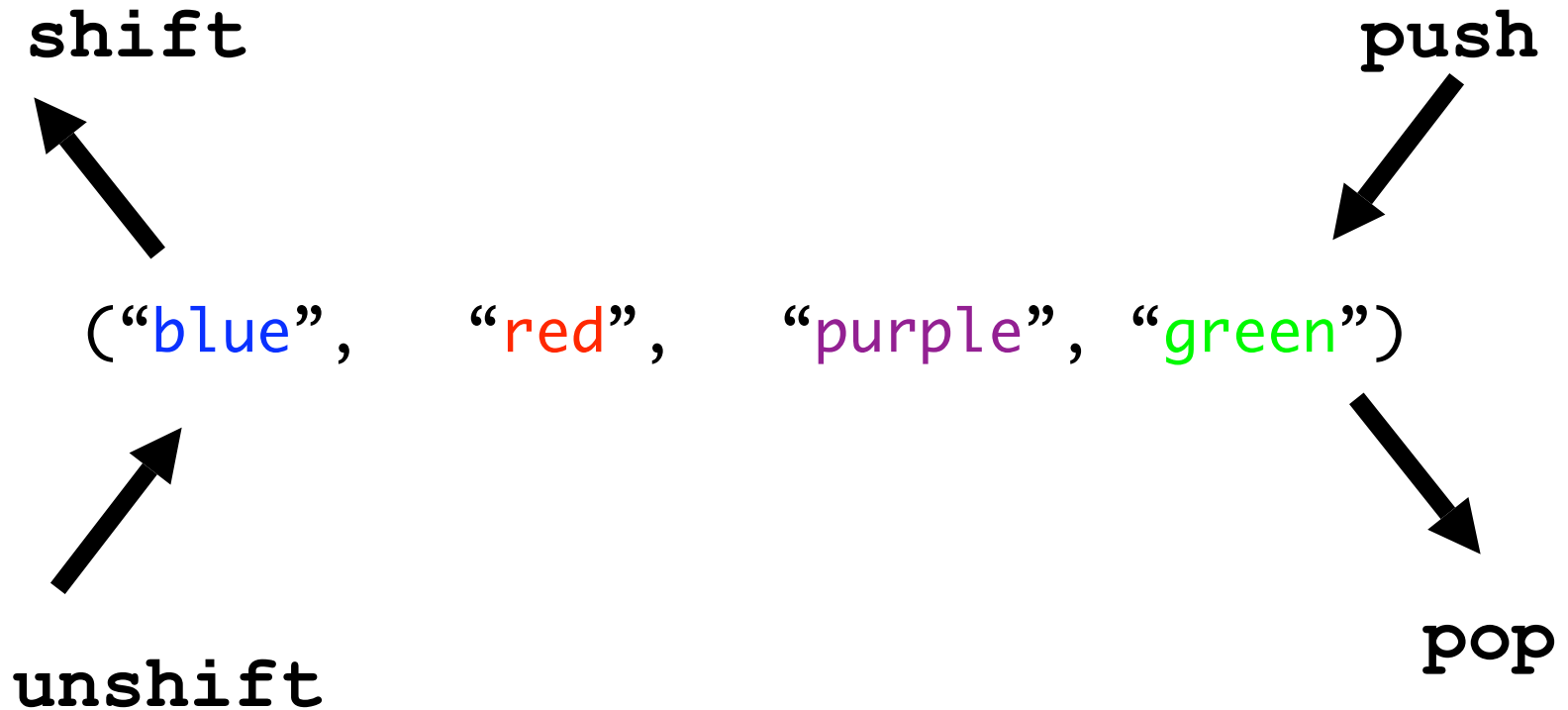


- Given this code as the first line:

```
@n = (9,20,4);
```

- Set \$x to 16, without any digits except 1! Use less than 22 characters.
- Whitespace doesn't count.

shift, unshift, push, pop



Shift, unshift

- Both operate on the front (or top, depending on how you think of it) of the array.

```
@colors = qw(blue red purple green);
```

```
$first = shift @colors;
```

```
# $first gets "blue", and @colors becomes  
# (red, purple, green)
```

```
unshift @colors, "orange";
```

```
# @colors becomes (orange, red, purple, green)
```

Pop, push

- Both operate on the back (or bottom, depending on how you think of it) of the array

```
@colors = qw(orange red purple green);
```

```
$last = pop @colors;
```

```
# $last gets 'green', and @colors
```

```
# becomes (orange, red, purple)
```

```
push @colors, 'blue';
```

```
# @colors is (orange, red, purple, blue)
```

FIFO vs LIFO

- How to build a First In First Out (FIFO or “queue”) data structure:

```
# use push and shift
@waitlist = ();
push(@waitlist, 'Sparky');
push(@waitlist, 'Rover');
$first_to_eat = shift @waitlist;
```

FIFO vs LIFO

- How to build a Last In First Out (stack) data structure

```
# use push and pop
@places = ();
push(@places, 'Seattle');
push(@places, 'Vancouver');
push(@places, 'Alaska');
$return_to = pop @places;
```

reverse

```
@animals = qw(elk antelope ant);  
@reversed = reverse @animals;
```

```
# @reversed is (ant antelope elk)  
# @animals HAS NOT CHANGED
```

```
@animals = reverse @animals;  
# this changes @animals itself
```

sort for an Array

- Sort returns a new sorted array

```
@stuff = qw(scissors paper rock);  
@sorted = sort @stuff;  
  
# @sorted is  
#      ('paper', 'rock', 'scissors')
```

Numerical Sort

- By default, the sort is in ASCII order (alphabetical)
- sort would do this to numbers:

```
@nums = sort (1,2,4,20,16);
```

```
# yields (1,16,2,20,4)
```

Numerical Sort

- This uses a subroutine reference, which we'll go into in more detail in the next course. For now, if you want numerical sorting just use this construct.
- You can also read about it with: `perldoc -f sort`

```
@nums = (1, 2, 4, 20, 16);  
@sorted = sort {$a <=> $b} @nums;
```

Group Challenge



- Given this code:

```
@c = qw(orange yellow green blue indigo violet red);
```

- Put @c in alphabetical order.
- Put @c in ROYGBIV (rainbow) order in as few statements as possible.
- Put @c in ROYGBIV order in one statement using only push, pop, shift and/or unshift!
- Put @c in ROYGBIV order in one statement, without other variables or shift/unshift/push/pop! (**impossible?**)

Split and Join

- split and join change strings to arrays
- and vice versa
- Very powerful
- We'll just learn the simple versions for now.



split

- split turns a string into an array

```
my @words = split(" ", "the quick brown fox");
```

```
@words is now  
("the", "quick", "brown", "fox");
```

split

- To convert a string to an array of characters, split on the empty string:

```
my @letters = split('', "Elbereth");
```

```
@letters is now
```

```
('E', 'l', 'b', 'e', 'r', 'e', 't', 'h');
```

join

- Join converts an array into a string
- Join is smart and only puts the joiner *between* the parts

```
my @guests = ("Finnegan", "Elijah", "Ian");  
push @guests, "Maya";  
  
print "Welcome to " . join(', ', @guests);
```

join

- Joining on empty string just concatenates the array:

```
my @list = ('a' .. 'z');  
  
print join('', @list);  
  
# prints 'abcdefghijklmnopqrstuvwxy'
```

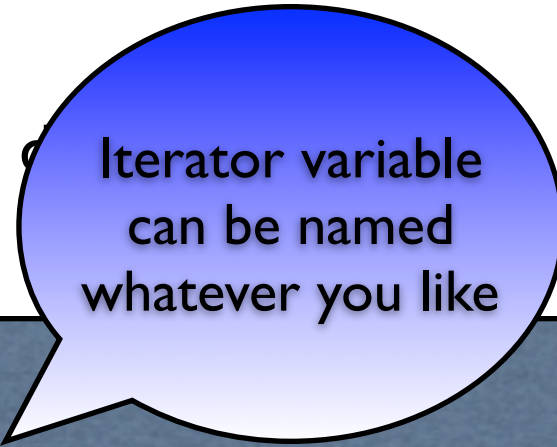
Working on an Array

- The "foreach" loop lets you do work with each member of an array:

```
foreach my $item (@array) {  
    print "$item\n";  
}
```

Working on an Array

- The "foreach" loop lets you iterate over each member of an array:



Iterator variable
can be named
whatever you like

```
foreach my $item (@array) {  
    print "$item\n";  
}
```

print for an Array

```
@animals = qw(dog cat goat);  
print @animals;  
# runs elements together -- "dogcatgoat"  
  
print "@animals";  
# separates elements with space chars  
# "dog cat goat"  
  
# see also:  join and foreach
```

Lab work: dead presidents

- Write a script to allow the user to look up Presidents of the USA. Limit it to the first 6 presidents:

```
'George Washington', 'John Adams', 'Thomas  
Jefferson', 'James Madison', 'James  
Monroe', 'John Quincy Adams'
```

```
: perl prez
```

```
Which president? 3
```

```
President 3 was Thomas Jefferson.
```

Lab work: smarter dead presidents

- Modify your president script to make sure the user types a number in the correct range.

```
% perl prez  
Which president? 12  
12 is out of range.  
I need a number between 1-6
```