Arrays

Sofia Robb

What is an Array?

- An array is a named list.
- What is a list?
  - (‘cat’, ‘dog’, ‘narwhal’)
- A named list:
  - @animals = (‘cat’, ‘dog’, ‘narwhal’);

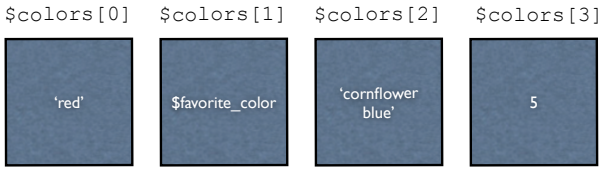
Arrays

- Arrays are denoted with '@' symbol

Arrays

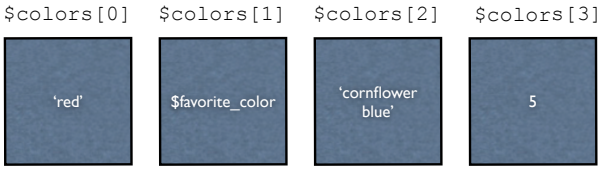
- Each element of an array is a scalar variable
  - number
  - letter
  - word
  - sentence
  - $scalar_variable

An array of colors.




```
my @colors = ('red', $favorite_color,
'cornflower blue', 5);
```


Accessing each element of an array by its index.




```
my @colors = ('red', $favorite_color,
'cornflower blue', 5);

my $first  = $color[0];
my $second = $color[1];
my $third  = $color[2];
my $last   = $color[-1];
```

Each element of the array is a scalar variable therefore we use the '$' when we refer to an individual element.

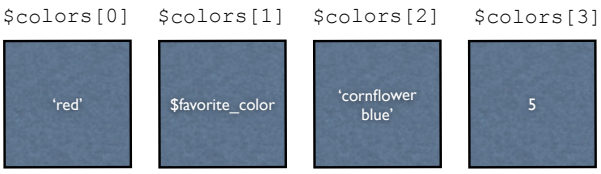
```
my $first  = $color[0];
my $second = $color[1];
my $third  = $color[2];
my $last   = $color[-1];
```


A common MISTAKE is to try to access an element in array context (meaning using the '@').

```
my @colors = ('red', $favorite_color,
'cornflower blue', 5);

This is wrong:
my $first  = @color[0];

This is correct:
my $first  = $color[0];
```

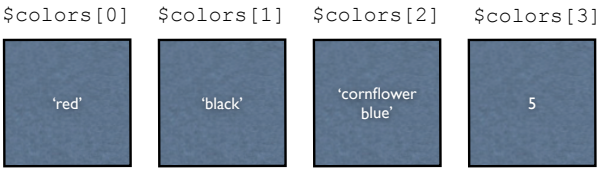
Changing values at a specific array index.

Original Array




```
$color[1] = 'black';
```

Edited Array



The value at position `$color[1]` is changed from $favorite_color to 'black'.

Calculate length of an array with `scalar`

```
my @colors = ('red', $favorite_color,
'cornflower blue', 5);

my $length = scalar @colors;
print "$length\n";
4

my $length = @colors;
print "$length\n";
4
```

Quick print of an array

```
my @colors = ('red', $favorite_color,
'cornflower blue', 5);

print "@colors";
red purple cornflower blue 5
```

When double quotes are used around "@array" in a print statement, the array elements are printed with a single space separating each element.

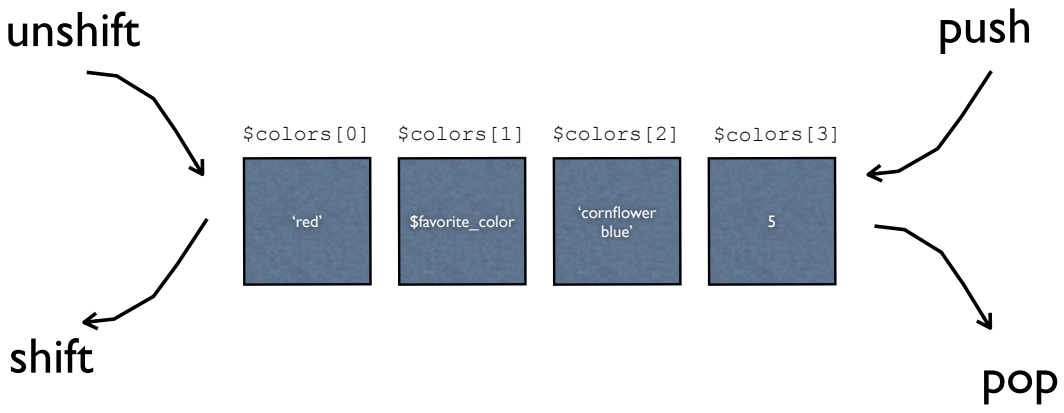
Converting an array to a string using `join()`

```
my $new_string = join(string , @array);

my @colors = ('red', $favorite_color,
'cornflower blue', 5);

my $new_string = join ('--' , @colors);
print "$new_string\n";
red--purple--cornflower blue--5
```

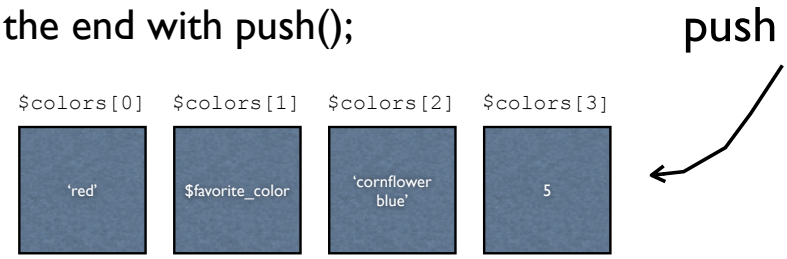
The `join()` function concatenates each element of the array with the provided 1st argument '--' into a string.

Arrays are Dynamic

Arrays can grow and shrink

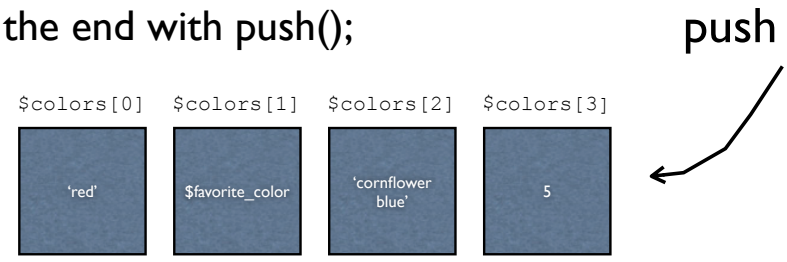


Add elements to the end with push();




```
push (@array, list of values);

#add one element to the end
push (@colors, 'black');

print join ('--', @colors) , "\n";
red--purple--cornflower blue--5--black
```

Add elements to the end with push();

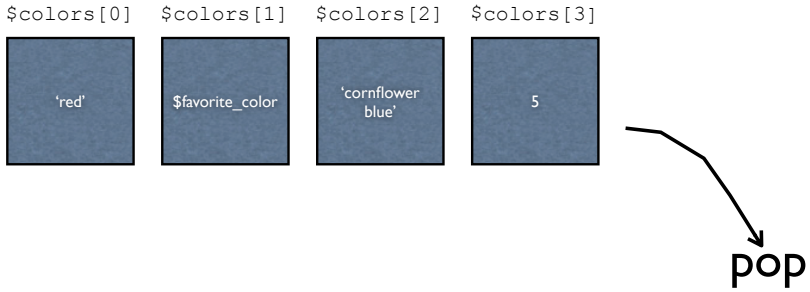



```
push (@array, list of values);

#add two elements to the end
push (@colors, 'black' , 'blue');

print join('--',@colors), "\n";
```
red--purple--cornflower blue--black--blue

Add elements to the end with push();


```
push (@array, list of values);

#add an array of elements
my @more_colors =
('yellow','pink','white','orange');

push (@colors, @more_colors);

print join('--',@colors) , "\n";
```
red--purple--cornflower blue--5--yellow--pink--white--orange

Remove an element from the end with pop();




```
my $last_element = pop @colors;

print "$last_element\n";
5
print join ('--', @colors) , "\n";
red--purple--cornflower blue
```


Remove an element from the beginning with shift();


```
my $first_element = shift(@colors);

print "$first_element\n";
red

print join ('--', @colors) , "\n";
purple--cornflower blue--5
```

Add elements to the beginning with unshift();


```
unshift (@array, list of values);

#add one element to the beginning
unshift (@colors, 'black');

print join ('--', @colors) , "\n";
black--red--purple--cornflower blue--5
```


Add elements to the beginning with unshift();


```
unshift (@array, list of values);

#add one element to the beginning
unshift (@colors, 'black');

print join ('--', @colors) , "\n";
black--red--purple--cornflower blue--5
```

Add elements to the beginning with unshift();


```
unshift (@array, list of values);

#add two elements to the beginning
unshift (@colors, 'black' , 'blue');

print join('--',@colors), "\n";
```
black--blue--red--purple--cornflower blue

Add elements to the beginning with unshift();


```
unshift (@array, list of values);

#add an array of elements to the beginning
my @more_colors =
('yellow','pink','white','orange');

unshift (@colors, @more_colors);

print join('--',@colors) , "\n";
```
yellow--pink--white--orange--red--purple--cornflower blue--5

Dynamic Arrays

Function	Meaning
push(@array, a list of values)	add value(s) to the end of the list
$popped_value = pop(@array)	remove a value from the end of the list
$shifted_value = shift(@array)	remove a value from the front of the list
unshift(@array, a list of values)	add value(s) to the front of the list
splice(...)	everything above and more!

Converting a string into an array

```
my @array = split(pattern , string);

my $string = "I do not like green eggs and ham";
my @words = split(' ',$string);

print join('--',@words),"\n";
```
I--do--not--like--green--eggs--and--ham

`split()` is splitting the string on ' ' (a single white space) into individual array elements.

Sorting the elements of an array

```
my @words = qw(I do not like green eggs and ham);

my @sorted_words = sort @words;

print join(@sorted_words),"\n";
I--and--do--eggs--green--ham--like--not
##ascii sort order. 0-9 then A-Z then a-z
```

The array sorts in ascii order not ABC order.

Sorting using the `cmp` operator

```
my @words = qw(I do not like green eggs and ham);

##sort {$a cmp $b} is default sort behavior
my @sorted_words = sort {$a cmp $b} @words;

print join(@sorted_words),"\n";
I--and--do--eggs--green--ham--like--not
```

The comparison operator and strings

```
my $x = 'sid';
my $y = 'nancy';
my $result = $x cmp $y;
```

$result is:

-1 if the ascii value of the left side is less than the right side
 0 if the ascii value of the left side equals the right side
+1 if the ascii value of the left side is greater than the right side

Reverse sorting of arrays using the `cmp` operator

```
my @words = qw(I do not like green eggs and ham);

my @sorted_words = sort {$b cmp $a} @words;

print join('--',@sorted_words),"\n";
not--like--ham--green--eggs--do--and--I
```

The comparison operator for numbers

```
my $x = 2;
my $y = 3.14;
my $result = $x <=> $y;
```

$result is:

-1 if the value of the left side is less than the right side
 0 if the value of the left side equals the right side
+1 if the value of the left side is greater than the right side

Numeric sorting of arrays using the <=> operator

```
my @numbers = (15,2,10,20,11,1);

## default sorting is ascii
my @sorted_numbers = sort @numbers;
print "@sorted_numbers\n";
1 10 11 15 2 20

@sorted_numbers = sort {$a <=> $b}@numbers;
print "@sorted_numbers\n";
1 2 10 11 15 20
```

With the <=> the numbers of the array sort by numeric value and not ascii value.

Using the `map` function with arrays

```
my @words = qw(I do not like green eggs and ham);

my @ABC_words = map { uc }@words;
print join('--',@ABC_words),"\n";
I--DO--NOT--LIKE--GREEN--EGGS--AND--HAM

my @sorted_words = sort (@ABC_words);
print join('--',@sorted_words),"\n";
AND--DO--EGGS--GREEN--HAM--I--LIKE--NOT
```

After converting to uppercase the array sorts in ABC order.

Accessing Each Element of an Array

- Loops
  - foreach
  - for
  - while

foreach loop

```
## iterate thru @array
foreach my $one_element(@array){
 ##do something to each $one_element
}
```


Iterating through an array with a foreach loop

```
my @words = qw(I do not like green eggs and ham);


foreach my $word (@words){
   print "$word\n";
}
```

I
do
not
like
green
eggs
and
ham

Sorting an array using `cmp` and iterating through each element

```
my @words = qw(I do not like green eggs and ham);

foreach my $word (sort {uc($a)cmp uc($b)}@words){
   print "$word\n";
}
```

and
do
eggs
green
ham
I
like
not

for loop iterations

```
for(initialization; test; increment){
   statements;
}
```

```
for (my $i=0; $i<5 ; $i++){
 print "$i\n";
}
```

0
1
2
3
4

for loop iterations

```
for (my $i=0; $i<5 ; $i++){
  print "$i\n";
}
```

```
0
1
2
3
4
```

$i	$i<5	print "$i\n";	$i++
0	yes	0	1
1	yes	1	2
2	yes	2	3
3	yes	3	4
4	yes	4	5
5	no		

while loop iterations

```
while(condition){
   statements;
}
```

```
my $i = 0;
while ($i<5){
   print "$i\n";
   $i++;
}
```

```
0
1
2
3
4
```

while loop iterations

```
my $i = 0;
while ($i<5){
   print "$i\n";
   $i++;
}
```

```
0
1
2
3
4
```

$i	$i<5	print "$i\n";	$i++
0	yes	0	1
1	yes	1	2
2	yes	2	3
3	yes	3	4
4	yes	4	5
5	no		

Loop Control: next

execution of next() will cause the loop to jump to the next iteration.

```
my @words = qw(I do not like green eggs and ham);

foreach my $word (sort {uc($a) cmp uc($b)}@words){
   next if $word eq 'and';
   print "$word\n";
}
```

```
do
eggs
green
ham
I
like
not
```

Loop Control: last

execution of last() will cause the loop to exit the loop.

```
my @words = qw(I do not like green eggs and ham);

foreach my $word (sort {uc($a) cmp uc($b)}@words){
   print "$word\n";
   last if $word eq 'and';
}
and
```


Example use of a loop to count the occurrences of a specific strings

```
my @seqs = qw(TTT CGG ATG TAA CCC ACC TGA);

my $count = 0;
foreach my $seq (@seqs){
  if ($seq eq 'TAA' or $seq eq 'TGA' or $seq eq 'TAG'){
    print "*\n";
  }else {
    $count++;
  }
}
print "$count non-stop codons\n";
```

@ARGV holds command line arguments

```
./sample_usr_input.pl 5 five
```

```
print "@ARGV\n";

print "\$ARGV[0]: $ARGV[0]\n";
print "\$ARGV[1]: $ARGV[1]\n";

my $arg1 = shift;
my $arg2 = shift;

print "arg1: $arg1\n";
print "arg2: $arg2\n";

print "\$ARGV[0]: $ARGV[0]\n";
print "\$ARGV[1]: $ARGV[1]\n";
```

@ARGV contains the 2 command line arguments 5 and five

The 2 command line arguments 5 and five are shifted off sequentially

@ARGV now is empty

```
5 five
$ARGV[0]: 5
$ARGV[1]: five
arg1: 5
arg2: five $ARGV
[0]:
$ARGV[1]:
```