Perl Scripting III

Arrays and Hashes

(Also known as

Data Structures)

Ed Lee & Suzi Lewis

Genome Informatics

Basic Syntax

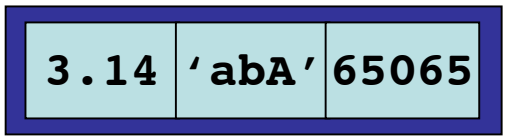
- In Perl the first character of the variable name determines how that variable will be interpreted when the code is run.
  - "$" indicates a "scalar" variable
  - "@" indicates an "array" variable
  - "%" indicates a "hash" variable
- You can have three variables with the same name
  - For example `$x`, `@x`, and `%x`
  - These represent three different things

An *Array* Is a *List* of Values

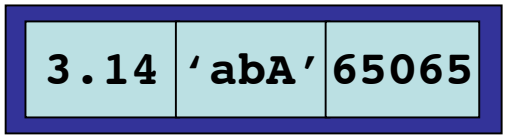
- For example, consider a list such as this
  - the number 3.14 as the first element
  - the string 'abA' as the second element
  - the number 65065 as the third element.
- How do you express this list in Perl?

"Literal Representation"

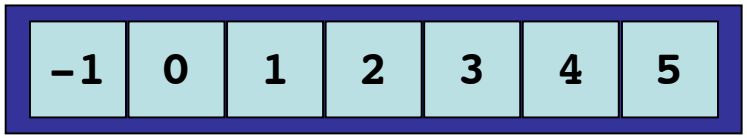
- Most simply
  - `my @array = (3.14, 'abA', 65065);`



- Or we can initialize from variables
  - `my $pi = 3.14;`
  - `my $s = 'abA';`
  - `my @array = ($pi, $s, 65065);`



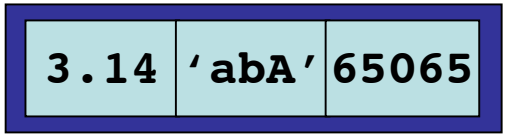
- We can also do integer ranges
  - `my @array = (-1..5); # shorthand for`

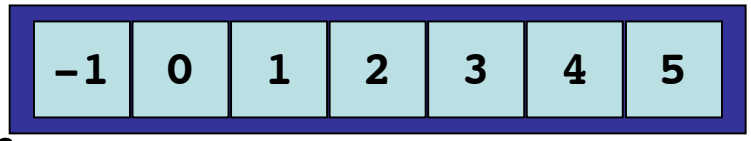


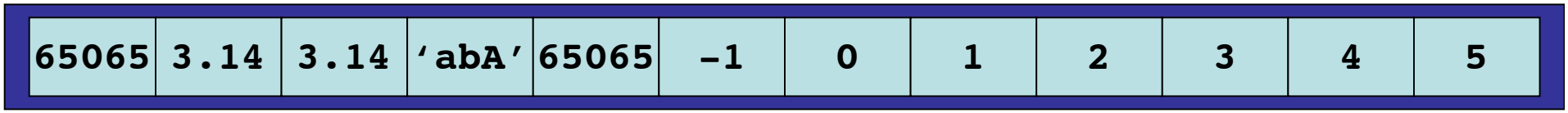
  - Counting down not allowed!

Array Variables and Assignment

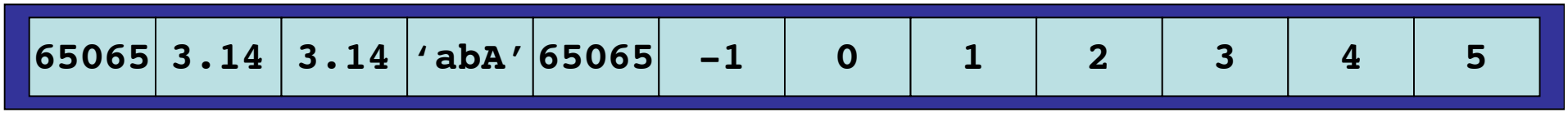
- `my $pi = 3.14;`
- `my $x = 65065;`
- `my @x = ($pi, 'abA', $x);`
- `my @y = (-1..5);`
- `my @z = ($x, $pi, @x, @y);`



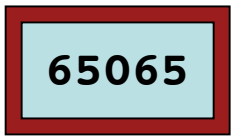


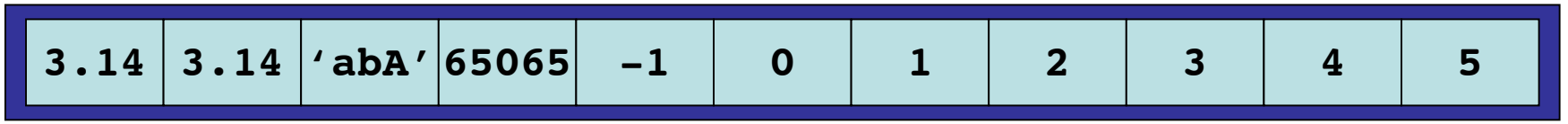


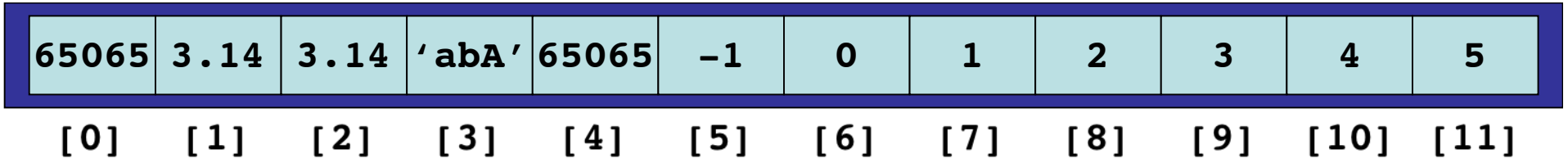
Array Variables and Assignment



- `my ($first, @rest) = @z;`

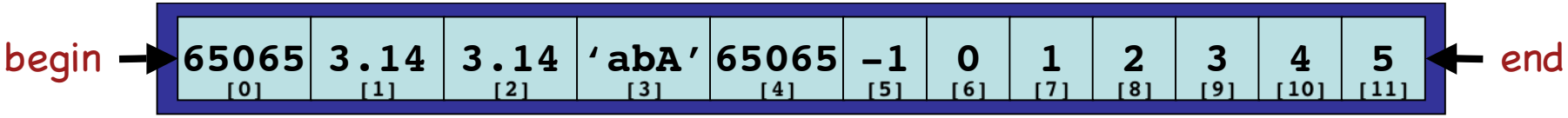






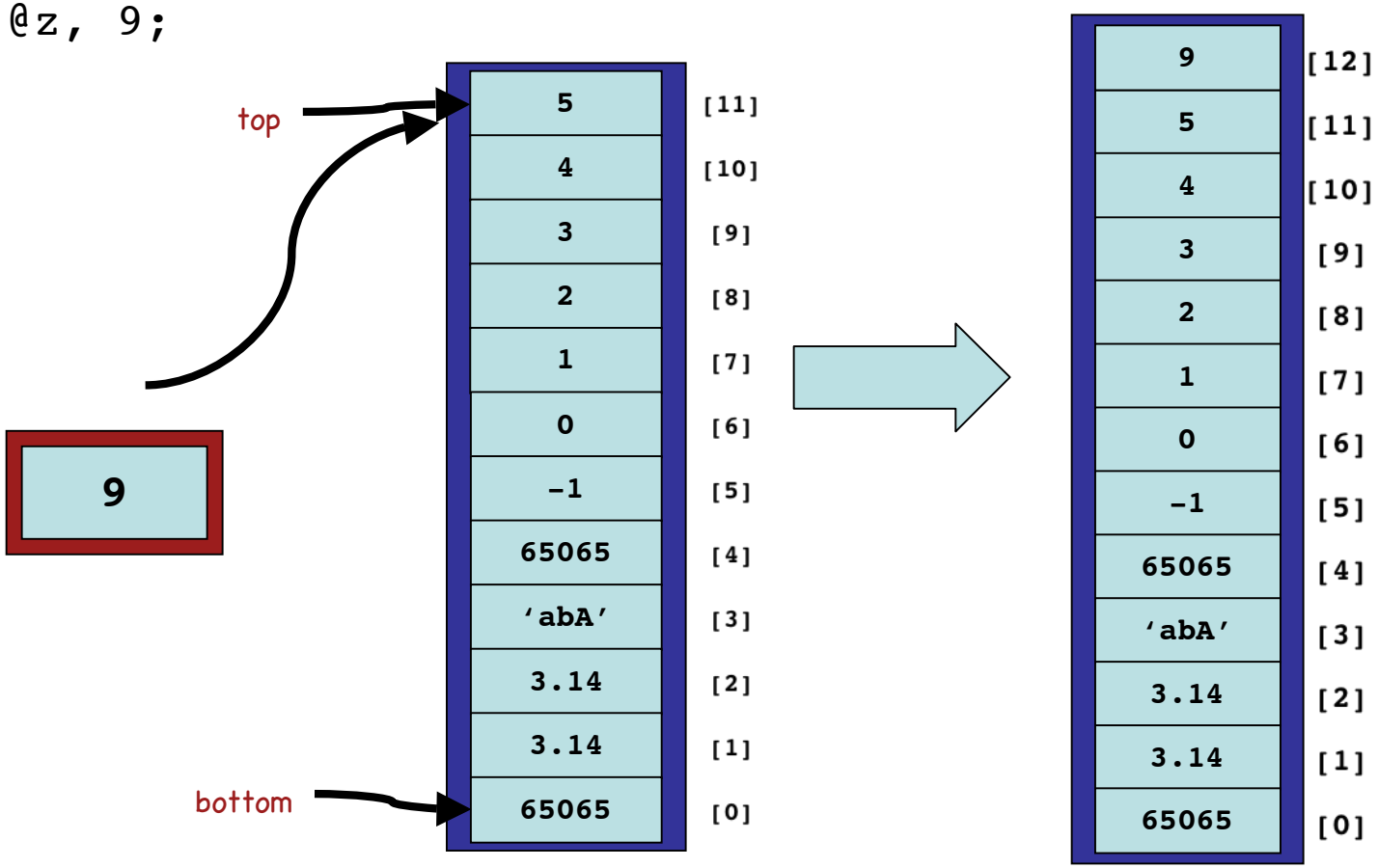
Getting at Array Elements

- `my $first = $z[0];` # 65065
- `$z[0] = 2; # assign a new value to the 1st item`
- `$first = $z[0];` # 2
- `my $max_index = $#z; # 11`
- `my $last = $z[$#z];` # 5



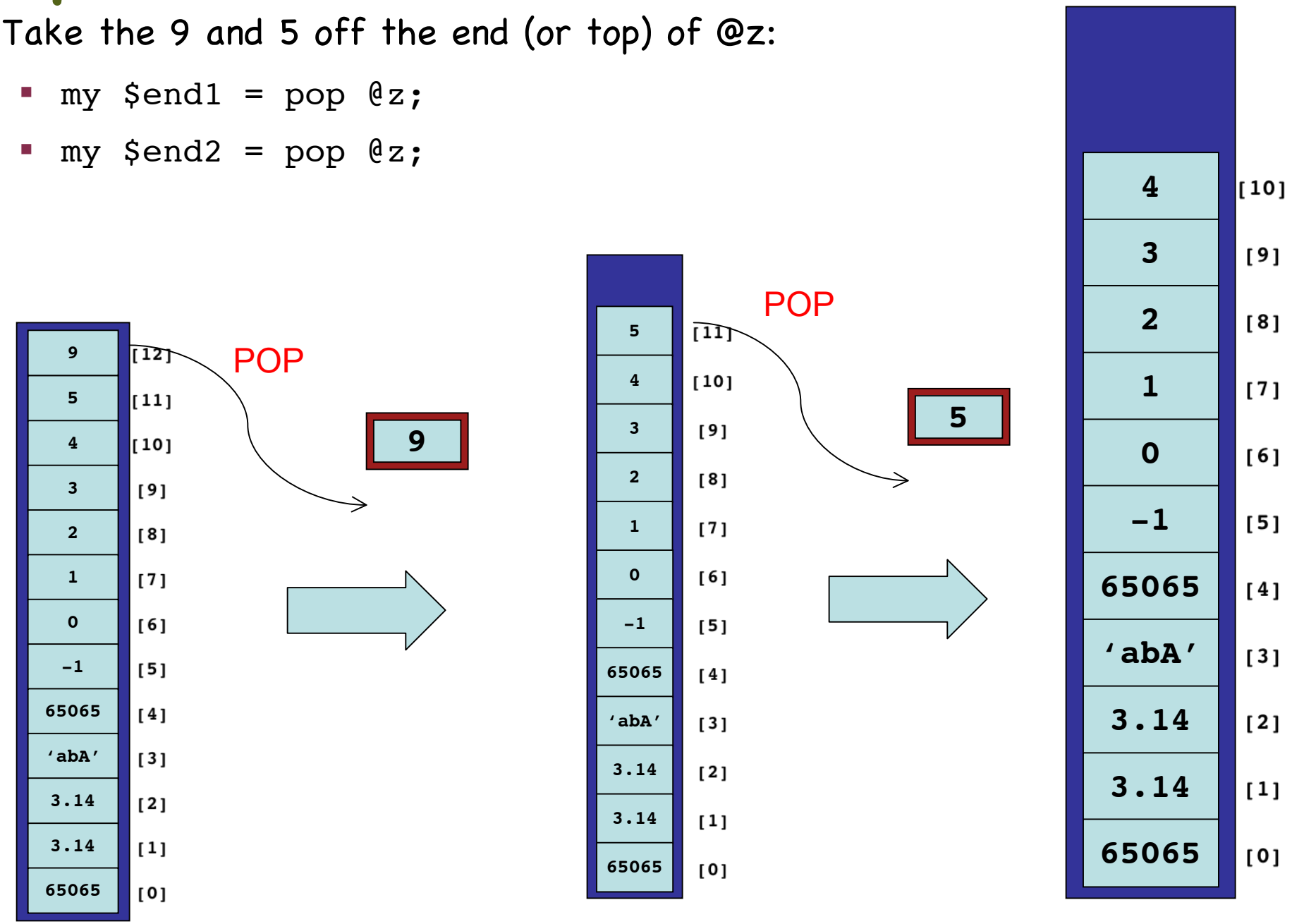
Push

- Add 9 to the end (or top) of @z;
  - `push @z, 9;`



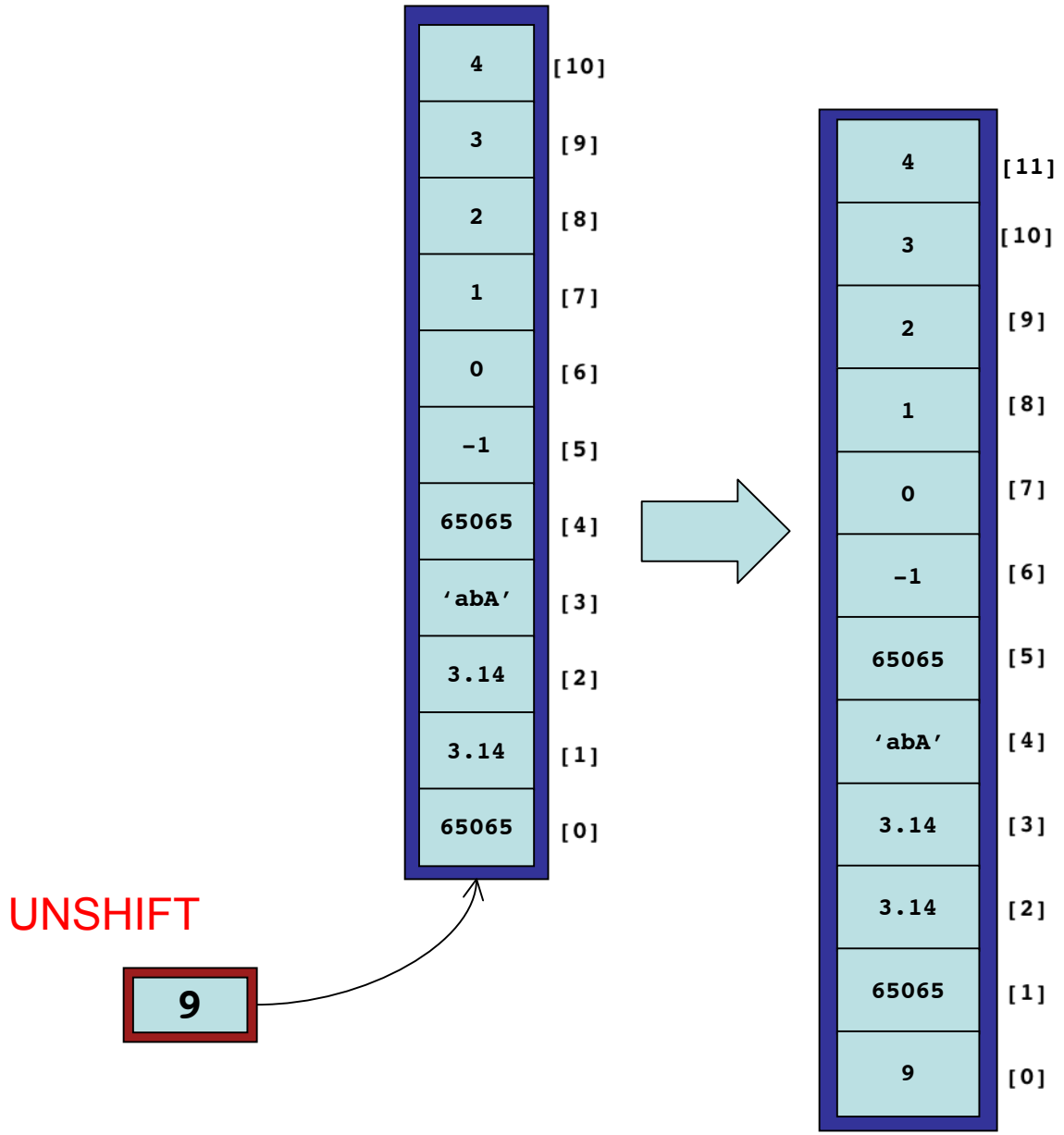
Pop

- Take the 9 and 5 off the end (or top) of @z:
  - `my $end1 = pop @z;`
  - `my $end2 = pop @z;`



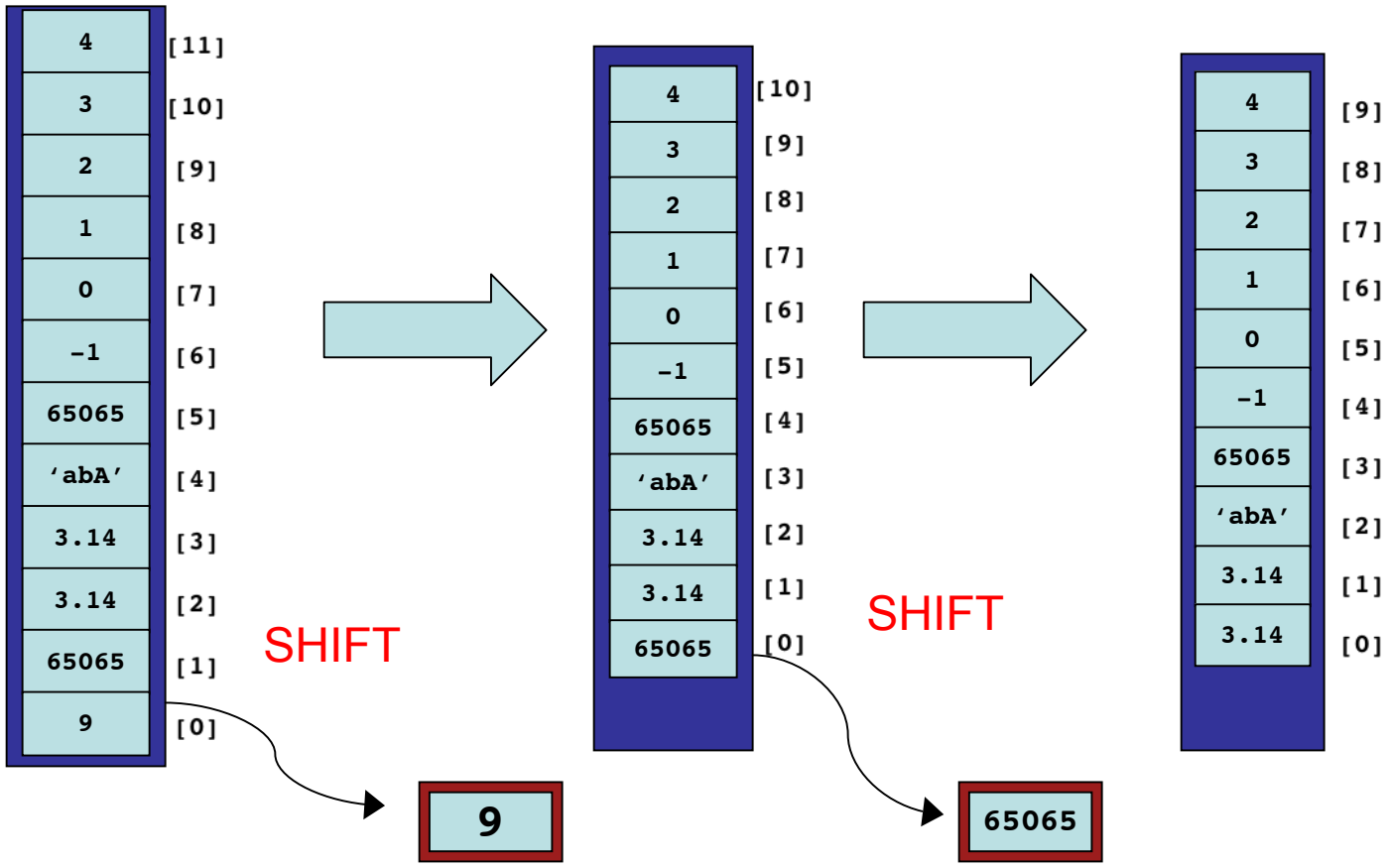
Unshift

- Add 9 to the beginning (or bottom) of @z;
  - `unshift @z, 9;`



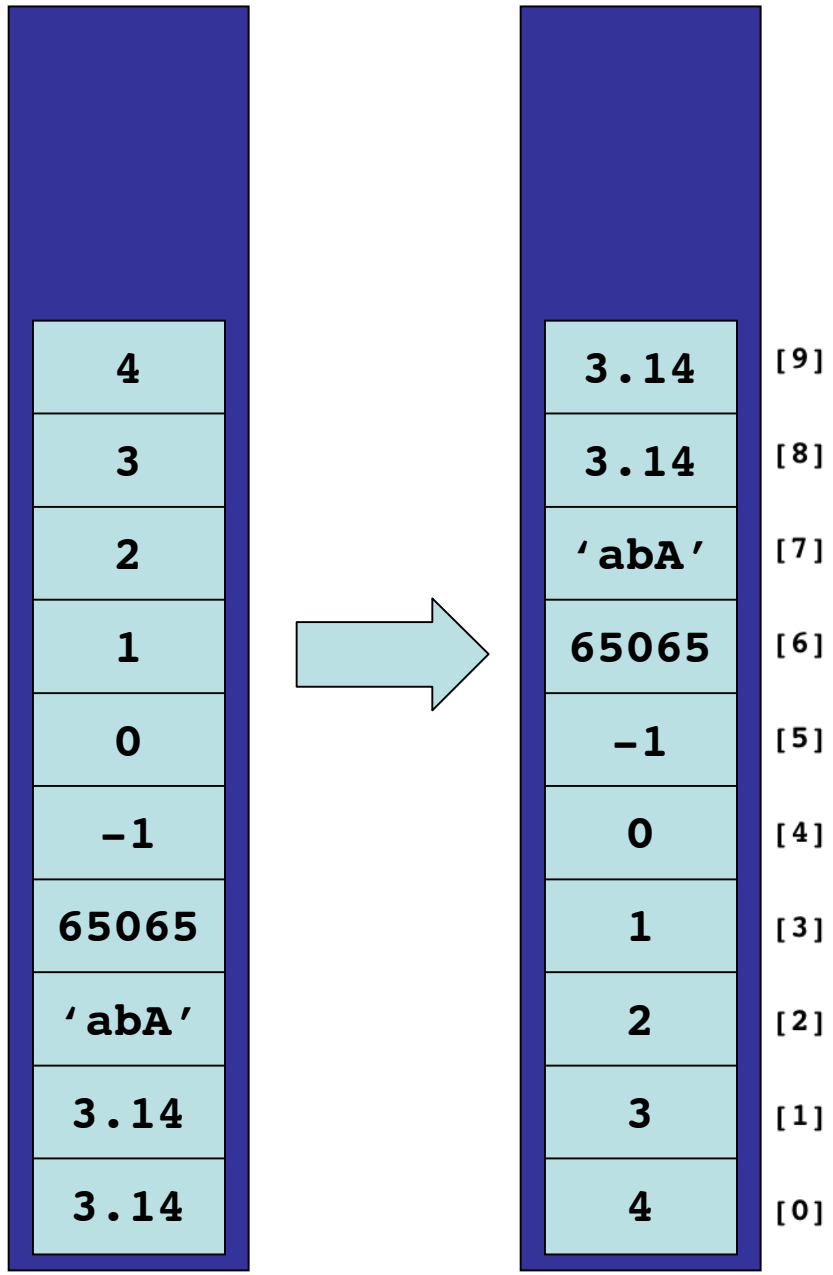
Shift

- Take 9 and then 65065 off the beginning of @z:
  - `my $b1 = shift @z;`
  - `my $b2 = shift @z;`



Reverse

- `my @zr = reverse @z;`



Array and Scalar Context

- The notion of array and scalar context is unique to Perl. Usually you can remain unaware of it, but it comes up in the **reverse** function.

```
print reverse 'abc';
```

```
abc
```

```
print reverse 'abc', 'def' , 'ghi' ;
```

```
ghidefabc
```

```
print scalar reverse 'abc';
```

```
cba
```

```
my $ba = reverse 'abc';
print $ba;
```

```
cba
```

Array and Scalar Context

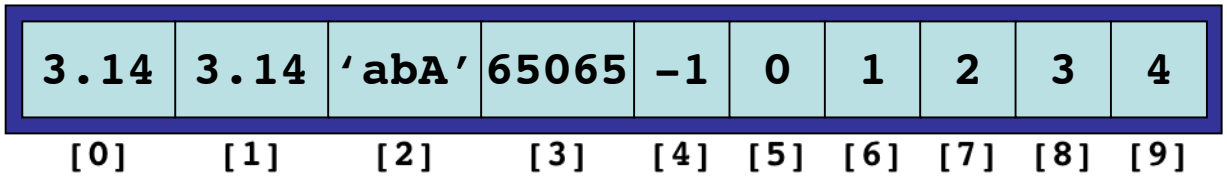
- The notion of array and scalar context can also be used to get the size of an array.

```
my @z = (1,2,3,4,5,6,7);
print scalar @z ," the number of elements in the
   array\n";
print $#z, ' this is max offset into scalar @z' , "\n";
```

7 the number of elements in the array

6 this is max offset into scalar @z

Iterating Through Array Contents



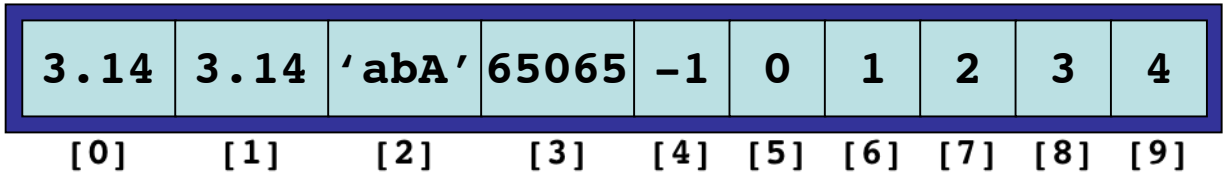
- Using a "foreach" loop

```
foreach my $array_value (@z) {
    print "$array_value\n";
}
```

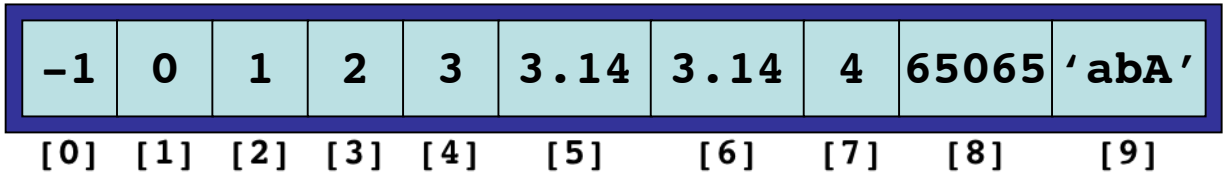
- Using a "for" loop

```
for (my $index = 0; $index < scalar(@z); ++$index) {
    my $array_value = $z[$index];
    print "$array_value\n";
}
```

Sorting



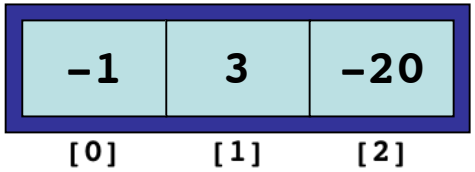
- Alphabetically:
  - `my @sortedArray = sort @z;`



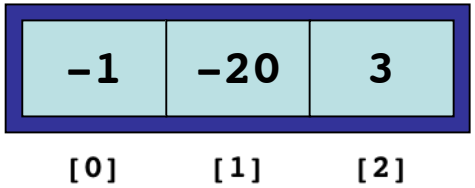
- This does exactly the same alphabetical sort
  - @sortedArray = sort {$a cmp $b} @z;

Sorting

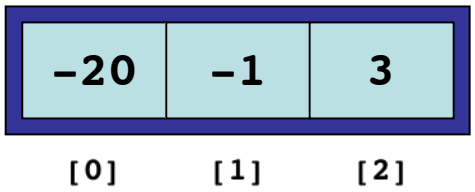
- An alphabetical sort (with only numbers in the array)
  - `my @numberArray = (-1, 3, -20);`



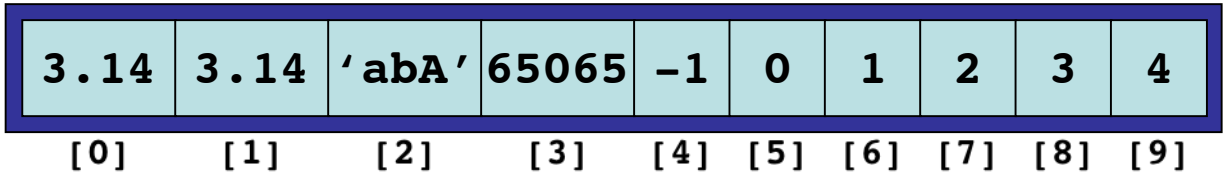
  - `my @sortedNums = sort @numberArray;`



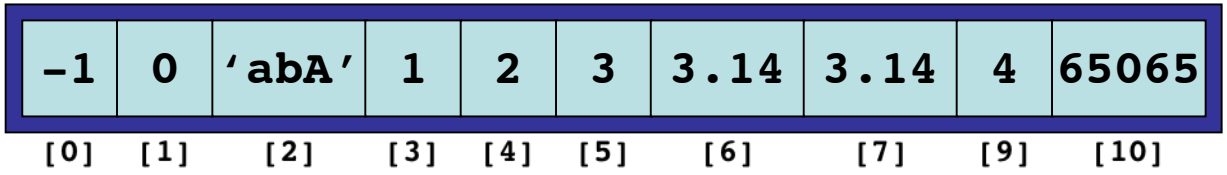
- Need a numerical sort to sort as numbers
  - my @sortedNums = sort {$a <=> $b} @numberArray;



Sorting



- What happens :
  - `@sortedArray = sort {$a <=> $b} @z;`
  - Argument "abA" isn't numeric in sort at arraySort.pl line 19.



Split and Join

- Split using a literal

```
my $string = "one,two,three";
my @array = split "," , $string;
print "@array" , ” - from array\n";

one two three - from array
```

- Join it up again

```
$string = join ‘:’, @array;
print $string , " - rejoined with colons\n";

one two three - from array
one:two:three - rejoined with colons
```

Split and Join

- Split using a regular expression

```
my $string = "one1two22three333fin";
my @array = split /\d+/ , $string;
print "@array" , "\n";

one two three fin
```

Swallowing Whole Files in a Single Gulp

- Read the file from stdin
  - `my @file = <>;`
- Eliminate newlines from each line
  - `chomp @file;`

A Hash Is a Lookup Table

- Hashes use a key to find an associated value.

```
my %translate; # the percent sign denotes a hash
$translate{'atg'} = 'M'; # codon is the key
$translate{'taa'} = '*'; # aa is the value
$translate{'ctt'} = 'K';  # lysine, oops
$translate{'ctt'} = 'L';  # leucine, fixed
print $translate{'atg'};
```

M

Initializing & Removing Key, Value Pairs

- Initializing From a List

```
%translate = ( 'atg' => 'M',
               'taa' => '*',
               'ctt' => 'L',
               'cct' => 'P' );
```

- Removing key-value pairs

```
delete $translate{'taa'};
```

Checking if a key exists

```
if (exists $translate{‘atg’}) {
   print “Methionine found in translation table\n”;
}
else {
   print “Methionine not found in translation table\n”;
}
if (exists $translate{‘ata’}) {
   print “Isoleucine found in translation table\n”;
}
else {
   print “Isoleucine not found in translation table\n”;
}
```

Methionine found in translation table
Isoleucine not found in translation table

Reaching into a hash

```
my @codons = keys %translate;
print "@codons" , " - all keys\n";
```

atg ctt taa - all keys

```
my @aa = values %translate;
print "@aa" , " - all values\n";
```

M L * - all values

Iterating Through Hash Contents

- First get all the keys from the hash

```
my @keys = keys %translate;
```

- Using a "foreach" loop

```
foreach my $key (@keys) {
    print "The AA code for ", $key, " is ", $translate{$key}, "\n";
}
```

- Using a "for" loop

```
for (my $index = 0; $index < scalar(@keys); ++$index) {
    my $key = $keys[$index];
    print "The AA code for ", $key, " is ", $translate{$key}, "\n";
}
```

Problem Sets

- Problem #1
  - Exercises 1-3, page 54, Learning Perl
- Problem #2
  - Exercises 1, page 105, Learning Perl
  - How the program (call it names.pl) in exercise 1 works:

```
$ names.pl fred
  flinstone
$ names.pl barney
  rubble
$ names.pl wilma
  flinstone
```

References

- Perl docs online perldoc.perl.org
- Learning Perl. Schwartz and Christiansen
  - Chapters 3 & 6
- Programming Perl. Wall, Christiansen and Schwartz
- Effective Perl Programming. Hall and Schwartz