CHAPTER 6: ARRAYS

One Dimensional Arrays

- Arrays are like rows of numbered compartments
- Arrays start counting their elements at the index zero
- The nth element can be found at index n - 1
- An array is one-dimensional when it has only one index or dimension
- To access an element in an array, use: array_name[index]

Arrays

- A data structure is any organized means of storage
- An array is a simple data structure, belonging to (instantiated from) the Array Class

Figure 6.1: An ordered list of variables

Example 6.2:

array = [1, 2, 3, 4, 5, 6, 7, 8, 9]

One Dimensional Arrays

- To create a new array, use:
  
  array_name = Array.new

Example 6.1:

arr = [73, 98, 86, 61, 96]

A simpler way to automatically create (instantiate) and initialize the same array (Example 6.2):

arr = [73, 98, 86, 61, 96]
One Dimensional Arrays

- To use the array, access `array_name[index]` as if it was a variable of the data type expected (Example 6.3)
  ```ruby
  arr = [5,6]
  arr[0] = arr[0] + 10
  puts arr[0]
  ```

- Arrays cluster multiple data items under one name

- Key advantage of using arrays: when they are used in conjunction with loops
  - Can use a variable for the index instead of literal numbers
    - You can change the index in every loop iteration and traverse through every element in the array

- To know when to stop traversing, get the number of elements in an array using: `arr.size`

- New programmers often make errors dealing with the bounds of an array
  - Basic rules for array bounds:
    - The first element in an array is at `index 0`
    - `arr.size` is not the highest indexed element
    - The last element in an array is at `arr.size - 1`

- To traverse an array using a while loop:
  - Initialize the index to 0
  - Increment it for every loop iteration
  - The condition is `index < arr.size`

Example 6.4:
```ruby
arr = [73, 98, 86, 61, 96]
index = 0
while (index < arr.size)
  puts arr[index]
  index = index + 1
end
```
One Dimensional Arrays

- Running the code gives the following output:
  - 73
  - 98
  - 86
  - 61
- That same array can be output with the code: `puts arr`

Example: Find the Max of an Array of Positive Numbers (Example 6.5)

```
# Initialize array and loop values
arr = [73, 98, 86, 61, 96]
index = 0
max = 0

# Loop over each element in arr
while (index < arr.size)
  if (arr[index] > max)
    # Update max
    max = arr[index]
  end
  index = index + 1
end

# Output calculated max
puts "Max ==> " + max.to_s
```

Summary

- An array is a data structure that stores multiple variables, belonging to the class Array
- Data stored in an array are accessed using numbers as an index starting at zero

Strings

- Strings are data structures that can be viewed as one dimensional arrays of character, BUT they are NOT arrays
- The most used string in programming books is "Hello World"
- It does not belong to the Class Array, but to the Class String
Strings

- Strings, however, look like arrays, so it is natural to have for them access mechanisms and methods similar to arrays

```
my_arr = Array.new
my_str = String.new
my_arr = [1,2,3,5,8]
my_arr.size # 5
my_arr[2..3] # [3,5]
my_arr[2,3] # [3,5,8]
```

```
my_str = "Hello World"
my_str.size # 11
my_str[2..3] # "ll"
my_str[2,3] # "llo"
my_str[2..4] # [3,5,8]
my_str[8..9] # "rl"
my_str[2,4] # [3,5,8]
my_str[8,9] # "rld"
```

Strings

- Strings, being elements (or objects) of the Class String, also have defined operations

```
"Hello" + " " + "World"
```

produces

```
"Hello World"
```

Strings and Arrays

- Arrays, being objects of the Class Array, also have defined operations, such as +, with a meaning similar to String

```
[1,2,3] + [3,5]
```

produces

```
[1,2,3,5]
```
Strings and Arrays

- What is the meaning of – for strings?
  “I am not” – “I am”
  Should it be “not”
  ???????

NO!!!!!!
The operation (method) -
Is NOT defined for the Class String

(c) 2012 Ophir Frieder et al

Strings and Arrays

- Note also the following
  3 * [1,2] is an error
  [1,2] * 3 is [1,2,1,2,1,2]
  3 * “ab” is an error
  “ab” * 3 is “ab ab ab”

(c) 2012 Ophir Frieder et al

Multi-Dimensional Arrays

- Arrays that have more than one dimension are called multidimensional arrays
- Ruby basically recognizes only one dimensional arrays, but it is very flexible
  - For Ruby, you must put an array inside an array
- A common type is the two-dimensional array, which is used to represent matrices and coordinate systems

(c) 2012 Ophir Frieder et al

Multi-Dimensional Arrays

- Consider the following set of grades:

<table>
<thead>
<tr>
<th>Name</th>
<th>Grades</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geraldo</td>
<td>73, 98, 86, 61, 96</td>
</tr>
<tr>
<td>Brittany</td>
<td>60, 90, 96, 92, 77</td>
</tr>
<tr>
<td>Michael</td>
<td>44, 50, 99, 65, 19</td>
</tr>
</tbody>
</table>

(c) 2012 Ophir Frieder et al
Multi-Dimensional Arrays

To represent the following data, use an array of arrays:

```ruby
arr = [ [73, 98, 86, 61, 96], # arr[0]
       [60, 90, 96, 92, 77], # arr[1]
```

To access an individual score, use:

```ruby
array[row][column]
```

To find Brittany’s score for her third exam, type:

```ruby
puts "Brittany's Third Exam: " + arr[1][2].to_s
```

Traversing a multidimensional array requires a nested loop for every additional dimension.

Example 6.6: Outputting Multidimensional Arrays

1. # Initialize array and loop values
2. arr = [[73, 98, 86, 61, 96],
3.         [60, 90, 96, 92, 77],
4.         [44, 50, 99, 65, 100]]
5. row = 0
6. column = 0
7. # Loop over each row
8. while (row < arr.size)
9.   # Loop over each column
10.  while (column < arr[row].size)
11.    # Print the item at position row x column
12.    puts arr[row][column]
13.    column = column + 1
14.  end
15.  # Reset column, advance row
16.  column = 0
17.  row = row + 1
18. end

You can also output everything using one line:

```ruby
puts arr
```

The only problem is that output will have no formatting.

Example 6.7: Modified Find the Max

1. # Initialize the array and index/score variables
2. arr = [[73, 98, 86, 61, 96],
3.         [60, 90, 96, 92, 77],
4.         [44, 50, 99, 65, 100]]
5. row = 0
6. column = 0
7. maxscore = 0
8. maxrow = 0
9. # for each row
10. while (row < arr.size)
11.   # for each column
12.     while (column < arr[row].size)
13.         # Find the item at position row x column
14.         puts arr[row][column]
15.         # Update score variables
16.         if (arr[row][column] > maxscore)
17.             maxrow = row
18.             maxscore = arr[row][column]
19.         end
20.     end
21.     column = column + 1
22. end
23. row = row + 1
24. end

You can also output everything using one line:

```ruby
puts arr
```

The only problem is that output will have no formatting.
Example 6.7 Cont’d

```ruby
21 column = column + 1
22 end
23 # reset column, increment row
24 column = 0
25 row = row + 1
26 end
27
28 # output name and high score information
29 if maxrow == 0
30 puts "Geraldo has the highest score."
31 elsif maxrow == 1
32 puts "Brittany has the highest score."
33 elsif maxrow == 2
34 puts "Michael has the highest score."
35 else
36 puts "Something didn’t work correctly."
37 end
38 puts "The high score was: " + maxscore.to_s
```

Output:

```
Michael has the highest score.
The high score was: 99.
```

Heterogeneous Arrays

- All our examples used **homogeneous arrays**
- In such arrays, all elements belong to the **same class**
- Ruby allows an **arbitrary mixing** of elements, creating arbitrary dimensioned heterogeneous arrays

Multi-Dimensional Arrays

```ruby
arr = Array.new
arr[0] = "Hi y'all"
arr[1] = 3.14159265
arr[2] = 17
arr[3] = [1, 2, 3]
```

```
arr is ["Hi y'all", 3.14159265, 17, [1, 2, 3]]
```
Summary

- **Arrays** are structures that use a table format to store variables
  - Data stored in an array are accessed using **numbers** as an index starting at zero
  - An array can have an infinite number of **dimensions** and can contain **heterogeneous data**
  - **Hashes** are like arrays, but can use any variable as a key

(c) 2012 Ophir Frieder et al