# CHAPTER 6:

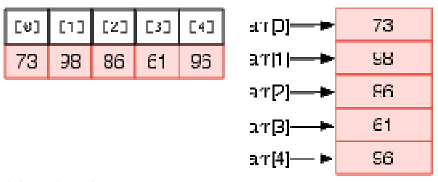
ARRAYS

Introduction to Computer Science Using Ruby

## 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



## One Dimensional Arrays

- Arrays are like rows of numbered compartments
- Arrays start counting their **elements** at the **index** zero
  - The $n^{th}$ 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]**

## One Dimensional Arrays

- To create a new array, use:

  **array_name = Array.new**

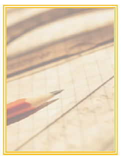
Example 6.1:

```
arr = Array.new
arr[0] = 73
arr[1] = 98
arr[2] = 86
arr[3] = 61
arr[4] = 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)

```
arr = [5,6]
arr[0] = arr[0] + 10
puts arr[0]
```

## One Dimensional Arrays

- 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

## One Dimensional Arrays

- 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`**

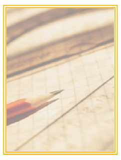
## One Dimensional Arrays

- 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:

```
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`

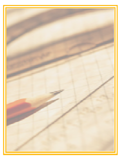
## Example: Find the Max of an Array of Positive Numbers (Example 6.5)

```ruby
# 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

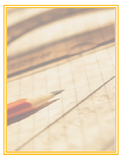
- **Strings** are data structures that can be viewed as one dimensional arrays of character, BUT they are **NOT** arrays
- The most used string in programming books is **"Hello World"**
- It does not belong to the Class Array, but to the **Class String**

## 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_str = “Hello World”
my_arr.size  #5             my_str.size  #11
my_arr.size  #3             my_str[2]  # “l”
my_arr[2..3] # [3,5]        my_str[2..3] # “ll”
my_arr[2,3]  # [3,5,8]      my_str[2,3]  # “llo”
my_arr[2..4] # [3,5,8]      my_str[8..9] # “rl”
my_arr[2,4]  # [3,5,8]      my_str[8,9]  # “rld”
```

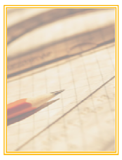
## Strings



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

**“Hello” + “ ” + “World”**
produces
**“Hello World”**

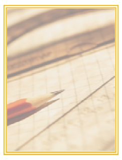
## Strings and Arrays



- 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
**[1,2,3,3,5]**

## Strings and Arrays

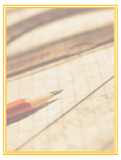


- Arrays, being **objects** of the Class **Array,** also have defined operations, such as **-** , which is a bit unusual

**[1,2,3] - [3,5]**
produces
**[1,2]**

**[3,5] – [1,2,3]**
produces
**[5]**

## Strings and Arrays

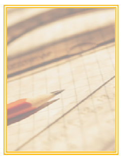


- What is the meaning of **–** for strings?

**" I am not" – "I am"**
Should it be **" not"**
**????????**

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

## 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 "**

## 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**

## Multi-Dimensional Arrays

- Consider the following set of grades:

| | |
|---|---|
| **Geraldo** | 73, 98, 86,61, 96 |
| **Brittany** | 60, 90, 96, 92, 77 |
| **Michael** | 44, 50, 99, 65, 19 |

## Multi-Dimensional Arrays

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

```
arr = [ [73,98,86,61,96], # arr[0]
        [60,90,96,92,77], # arr[1]
        [44,50,99,65,100] ] # arr[2]
```

## Multi-Dimensional Arrays

- To access an individual score, use:
  `array[row][column]`
- To find Brittany's score for her third exam, type:
  `puts “Brittany’s Third Exam: ” + arr[1][2].to_s`
  (Note the use of “ ” to allow the ’s)
- The output should be: `Brittany's Third Exam: 96`
- Traversing a multidimensional array requires a **nested loop** for every additional dimension

## Example 6.6: Outputting Multidimensional Arrays

```
# Initialize array and loop values
arr = [[73, 98, 86, 61, 96],
       [60, 90, 96, 92, 77],
       [44, 50, 99, 65, 100]]
row = 0
column = 0

# Loop over each row
while (row < arr.size)
    puts "Row: " + row.to_s
    # Loop over each column
    while (column < arr[row].size)
              # Print the item at position row x column
              puts arr[row][column]
              column = column + 1
    end
    # Reset column, advance row
    column = 0
    row = row + 1
end
```

You can also output everything using **one line:**
`puts arr`
The only problem is that output will have **no formatting**

## Example 6.7: Modified Find the Max

```
# initialize the array and index/score variables
arr = [[73, 98, 86, 61, 96],
       [60, 90, 96, 92, 77],
       [44, 50, 99, 65, 10]]

row = 0
column = 0
maxscore = 0
maxrow = 0

# for each row
while (row < arr.size)
   # for each column
   while (column < arr[row].size)
      # update score variables
      if (arr[row][column] > maxscore)
         maxrow = row
         maxscore = arr[row][column]
      end
      # increment column
```

## Example 6.7 Cont'd

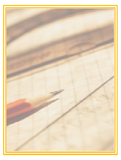
```
      column = column + 1
   end
   # reset column, increment row
   column = 0
   row = row + 1
end

# output name and high score information
if maxrow == 0
   puts "Geraldo has the highest score."
elsif maxrow == 1
   puts "Brittany has the highest score."
elsif maxrow == 2
   puts "Michael has the highest score."
else
   puts "Something didn't work correctly."
end
puts "The high score was: " + maxscore.to_s
```

## Output:

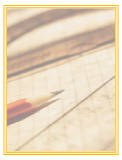
```
Michael has the highest score.
The high score was: 99.
```

## Heterogeneous Arrays



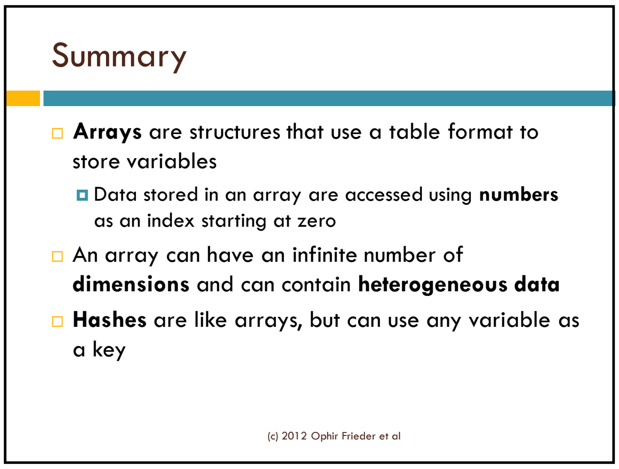
- All our examples used **homogeneous arrays**
- In such arrays, all elements belong to the **same class**
- Ruby allows an **arbitrary mixing** of elements, creating arbitrary dimensioned heterogeneous arrays

## Multi-Dimensional Arrays



```
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