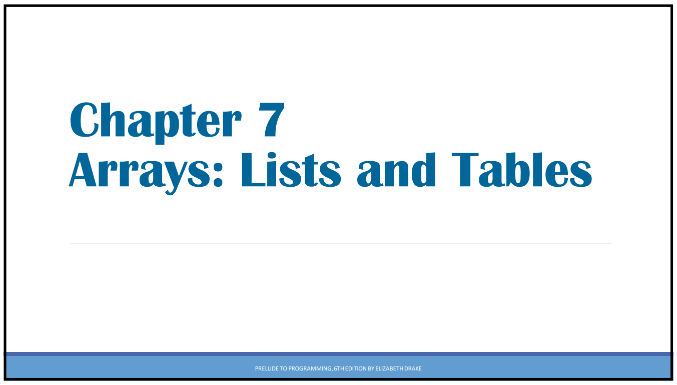
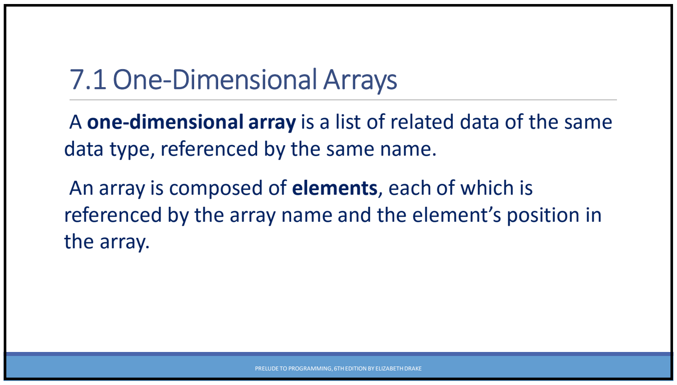


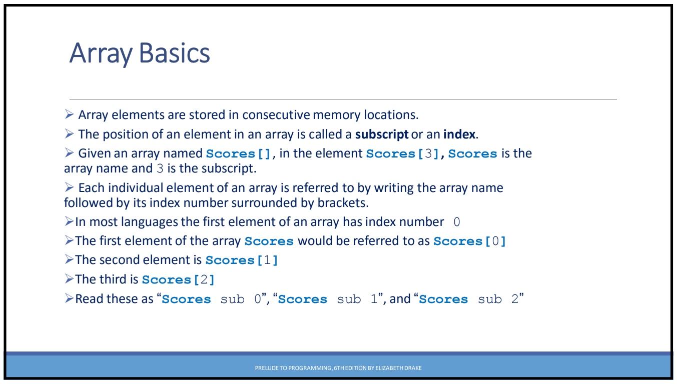
# Chapter 7
# Arrays: Lists and Tables



## 7.1 One-Dimensional Arrays

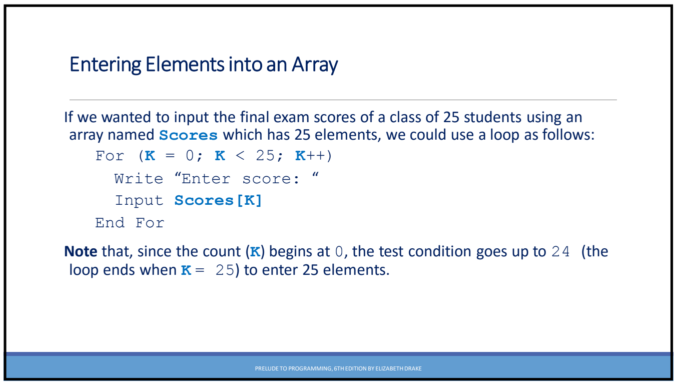
A **one-dimensional array** is a list of related data of the same data type, referenced by the same name.

An array is composed of **elements**, each of which is referenced by the array name and the element's position in the array.



## Array Basics

- ➢ Array elements are stored in consecutive memory locations.
- ➢ The position of an element in an array is called a **subscript** or an **index**.
- ➢ Given an array named `Scores[]`, in the element `Scores[3]`, `Scores` is the array name and `3` is the subscript.
- ➢ Each individual element of an array is referred to by writing the array name followed by its index number surrounded by brackets.
- ➢In most languages the first element of an array has index number `0`
- ➢The first element of the array `Scores` would be referred to as `Scores[0]`
- ➢The second element is `Scores[1]`
- ➢The third is `Scores[2]`
- ➢Read these as "`Scores sub 0`", "`Scores sub 1`", and "`Scores sub 2`"

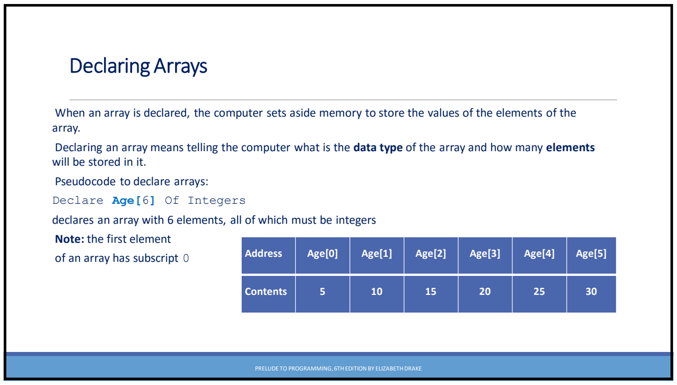


## Entering Elements into an Array

If we wanted to input the final exam scores of a class of 25 students using an array named **Scores** which has 25 elements, we could use a loop as follows:

```
For (K = 0; K < 25; K++)
  Write "Enter score: "
  Input Scores[K]
End For
```

**Note** that, since the count (K) begins at 0, the test condition goes up to 24 (the loop ends when K = 25) to enter 25 elements.



## Declaring Arrays

When an array is declared, the computer sets aside memory to store the values of the elements of the array.

Declaring an array means telling the computer what is the **data type** of the array and how many **elements** will be stored in it.

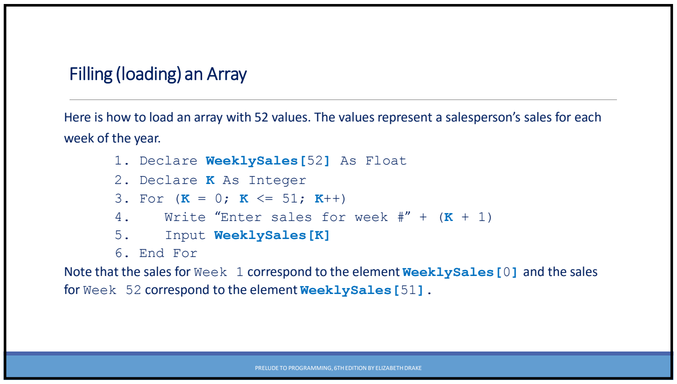
Pseudocode to declare arrays:

```
Declare Age[6] Of Integers
```

declares an array with 6 elements, all of which must be integers

**Note:** the first element of an array has subscript 0

| Address | Age[0] | Age[1] | Age[2] | Age[3] | Age[4] | Age[5] |
|---|---|---|---|---|---|---|
| Contents | 5 | 10 | 15 | 20 | 25 | 30 |

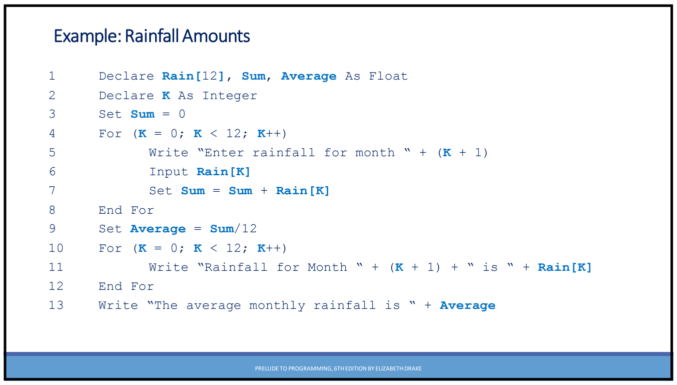


## Filling (loading) an Array

Here is how to load an array with 52 values. The values represent a salesperson's sales for each week of the year.

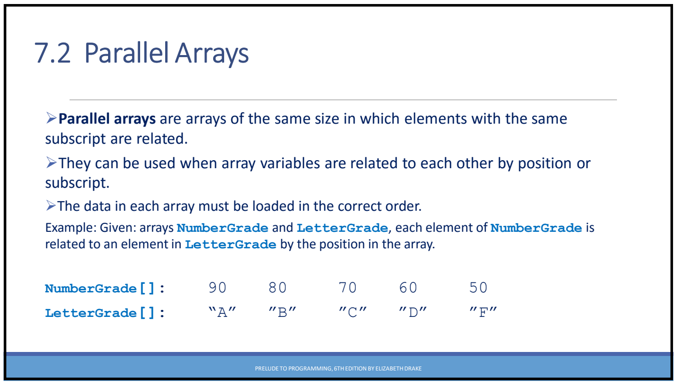
```
Declare WeeklySales[52] As Float
Declare K As Integer
For (K = 0; K <= 51; K++)
   Write "Enter sales for week #" + (K + 1)
   Input WeeklySales[K]
End For
```

Note that the sales for Week 1 correspond to the element WeeklySales[0] and the sales for Week 52 correspond to the element WeeklySales[51].



## Example: Rainfall Amounts

```
Declare Rain[12], Sum, Average As Float
Declare K As Integer
Set Sum = 0
For (K = 0; K < 12; K++)
      Write "Enter rainfall for month " + (K + 1)
      Input Rain[K]
      Set Sum = Sum + Rain[K]
End For
Set Average = Sum/12
For (K = 0; K < 12; K++)
      Write "Rainfall for Month " + (K + 1) + " is " + Rain[K]
End For
Write "The average monthly rainfall is " + Average
```

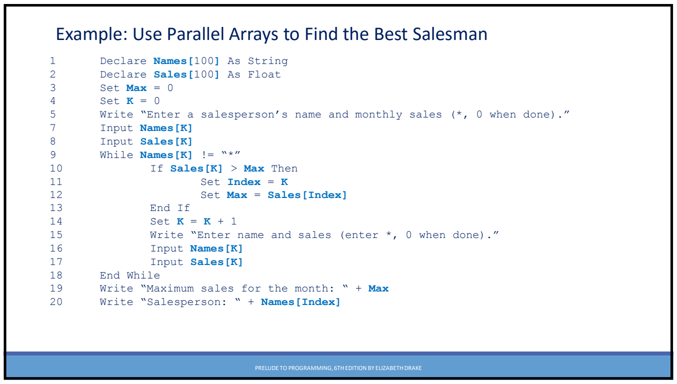


## 7.2 Parallel Arrays

- ➢**Parallel arrays** are arrays of the same size in which elements with the same subscript are related.
- ➢They can be used when array variables are related to each other by position or subscript.
- ➢The data in each array must be loaded in the correct order.

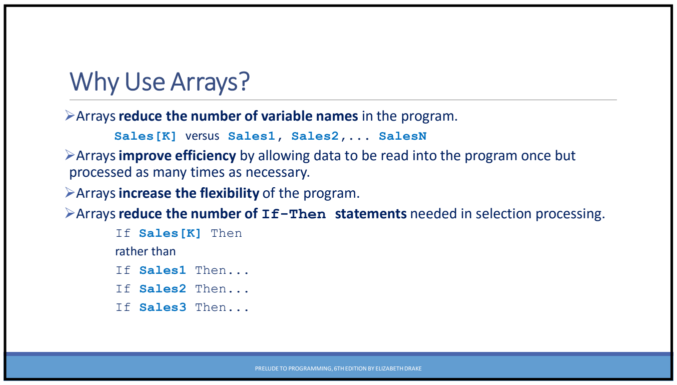
Example: Given: arrays `NumberGrade` and `LetterGrade`, each element of `NumberGrade` is related to an element in `LetterGrade` by the position in the array.

| | | | | | |
|---|---|---|---|---|---|
| NumberGrade[]: | 90 | 80 | 70 | 60 | 50 |
| LetterGrade[]: | "A" | "B" | "C" | "D" | "F" |



## Example: Use Parallel Arrays to Find the Best Salesman

```
Declare Names[100] As String
Declare Sales[100] As Float
Set Max = 0
Set K = 0
Write "Enter a salesperson's name and monthly sales (*, 0 when done)."
Input Names[K]
Input Sales[K]
While Names[K] != "*"
      If Sales[K] > Max Then
            Set Index = K
            Set Max = Sales[Index]
      End If
      Set K = K + 1
      Write "Enter name and sales (enter *, 0 when done)."
      Input Names[K]
      Input Sales[K]
End While
Write "Maximum sales for the month: " + Max
Write "Salesperson: " + Names[Index]
```



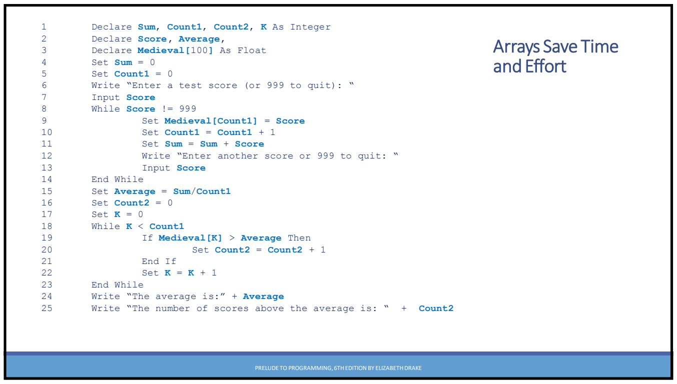
## Why Use Arrays?

- Arrays **reduce the number of variable names** in the program.

  **Sales[K]** versus **Sales1, Sales2,... SalesN**

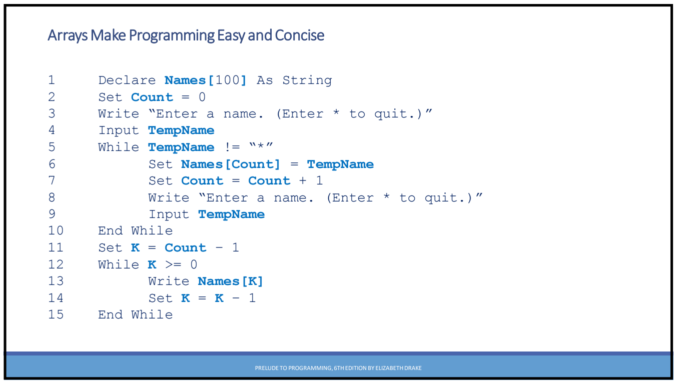
- Arrays **improve efficiency** by allowing data to be read into the program once but processed as many times as necessary.
- Arrays **increase the flexibility** of the program.
- Arrays **reduce the number of If-Then statements** needed in selection processing.

```
If Sales[K] Then
rather than
If Sales1 Then...
If Sales2 Then...
If Sales3 Then...
```



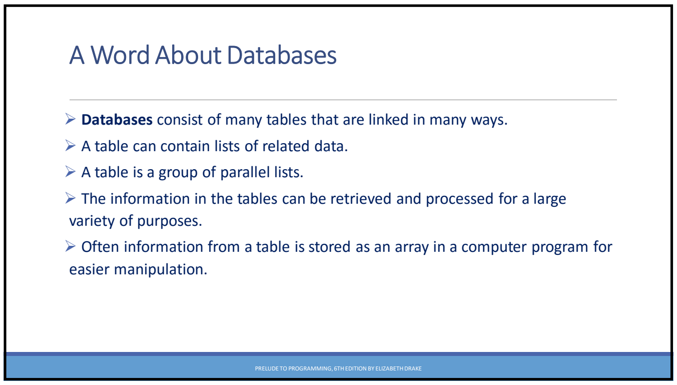
## Arrays Save Time and Effort

```
Declare Sum, Count1, Count2, K As Integer
Declare Score, Average,
Declare Medieval[100] As Float
Set Sum = 0
Set Count1 = 0
Write "Enter a test score (or 999 to quit): "
Input Score
While Score != 999
    Set Medieval[Count1] = Score
    Set Count1 = Count1 + 1
    Set Sum = Sum + Score
    Write "Enter another score or 999 to quit: "
    Input Score
End While
Set Average = Sum/Count1
Set Count2 = 0
Set K = 0
While K < Count1
    If Medieval[K] > Average Then
        Set Count2 = Count2 + 1
    End If
    Set K = K + 1
End While
Write "The average is:" + Average
Write "The number of scores above the average is: "  +  Count2
```



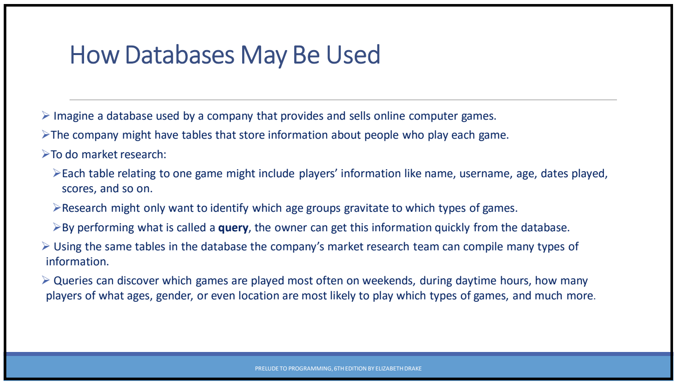
## Arrays Make Programming Easy and Concise

```
Declare Names[100] As String
Set Count = 0
Write "Enter a name. (Enter * to quit.)"
Input TempName
While TempName != "*"
    Set Names[Count] = TempName
    Set Count = Count + 1
    Write "Enter a name. (Enter * to quit.)"
    Input TempName
End While
Set K = Count – 1
While K >= 0
    Write Names[K]
    Set K = K – 1
End While
```



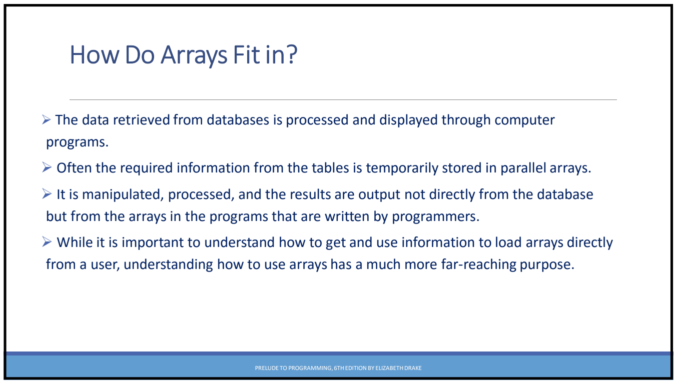
## A Word About Databases

- **Databases** consist of many tables that are linked in many ways.
- A table can contain lists of related data.
- A table is a group of parallel lists.
- The information in the tables can be retrieved and processed for a large variety of purposes.
- Often information from a table is stored as an array in a computer program for easier manipulation.



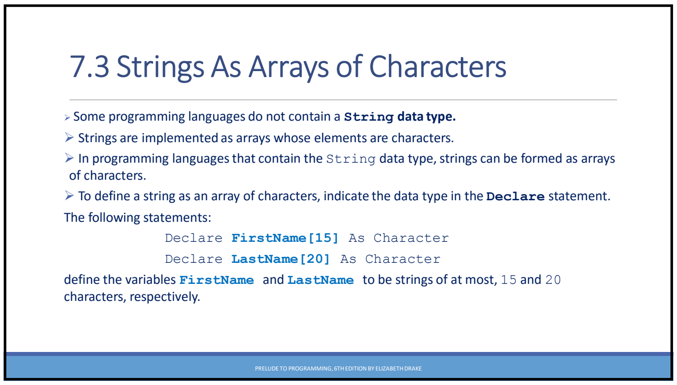
## How Databases May Be Used

- Imagine a database used by a company that provides and sells online computer games.
- The company might have tables that store information about people who play each game.
- To do market research:
  - Each table relating to one game might include players' information like name, username, age, dates played, scores, and so on.
  - Research might only want to identify which age groups gravitate to which types of games.
  - By performing what is called a **query**, the owner can get this information quickly from the database.
- Using the same tables in the database the company's market research team can compile many types of information.
- Queries can discover which games are played most often on weekends, during daytime hours, how many players of what ages, gender, or even location are most likely to play which types of games, and much more.



## How Do Arrays Fit in?

- The data retrieved from databases is processed and displayed through computer programs.
- Often the required information from the tables is temporarily stored in parallel arrays.
- It is manipulated, processed, and the results are output not directly from the database but from the arrays in the programs that are written by programmers.
- While it is important to understand how to get and use information to load arrays directly from a user, understanding how to use arrays has a much more far-reaching purpose.



## 7.3 Strings As Arrays of Characters

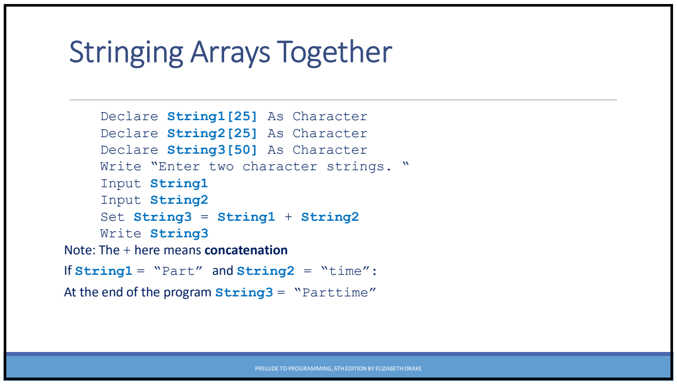
- Some programming languages do not contain a `String` **data type.**
- Strings are implemented as arrays whose elements are characters.
- In programming languages that contain the `String` data type, strings can be formed as arrays of characters.
- To define a string as an array of characters, indicate the data type in the `Declare` statement.

The following statements:

```
            Declare FirstName[15] As Character
            Declare LastName[20] As Character
```

define the variables `FirstName` and `LastName` to be strings of at most, 15 and 20 characters, respectively.

PRELUDE TO PROGRAMMING, 6TH EDITION BY ELIZABETH DRAKE



## Stringing Arrays Together

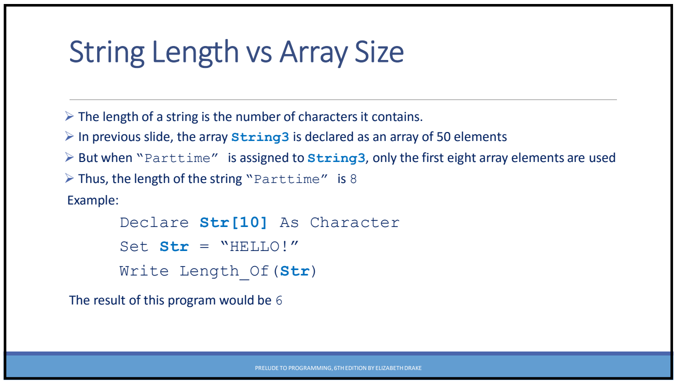
```
    Declare String1[25] As Character
    Declare String2[25] As Character
    Declare String3[50] As Character
    Write "Enter two character strings. "
    Input String1
    Input String2
    Set String3 = String1 + String2
    Write String3
```

Note: The + here means **concatenation**

If `String1` = "Part" and `String2` = "time":

At the end of the program `String3` = "Parttime"

PRELUDE TO PROGRAMMING, 6TH EDITION BY ELIZABETH DRAKE



## String Length vs Array Size

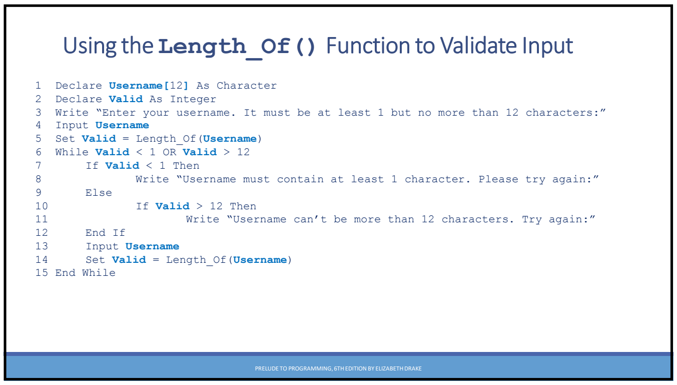
- The length of a string is the number of characters it contains.
- In previous slide, the array `String3` is declared as an array of 50 elements
- But when "Parttime" is assigned to `String3`, only the first eight array elements are used
- Thus, the length of the string "Parttime" is 8

Example:

```
      Declare Str[10] As Character
      Set Str = "HELLO!"
      Write Length_Of(Str)
```

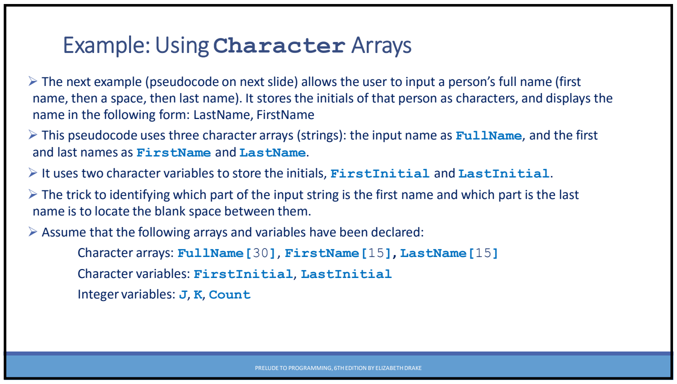
The result of this program would be 6

PRELUDE TO PROGRAMMING, 6TH EDITION BY ELIZABETH DRAKE



## Using the **Length_Of()** Function to Validate Input

```
Declare Username[12] As Character
Declare Valid As Integer
Write "Enter your username. It must be at least 1 but no more than 12 characters:"
Input Username
Set Valid = Length_Of(Username)
While Valid < 1 OR Valid > 12
     If Valid < 1 Then
             Write "Username must contain at least 1 character. Please try again:"
     Else
             If Valid > 12 Then
                     Write "Username can't be more than 12 characters. Try again:"
     End If
     Input Username
     Set Valid = Length_Of(Username)
End While
```



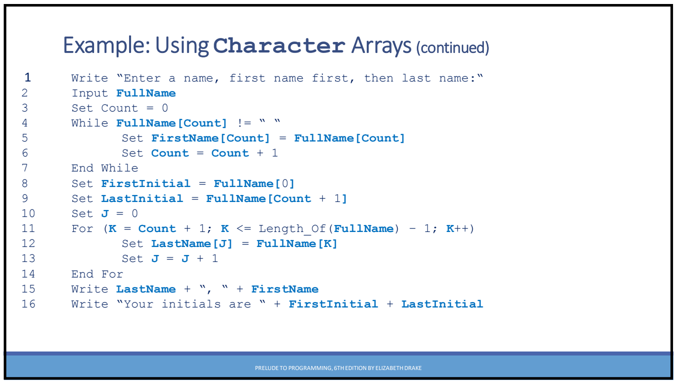
## Example: Using **Character** Arrays

- ➢ The next example (pseudocode on next slide) allows the user to input a person's full name (first name, then a space, then last name). It stores the initials of that person as characters, and displays the name in the following form: LastName, FirstName
- ➢ This pseudocode uses three character arrays (strings): the input name as **FullName**, and the first and last names as **FirstName** and **LastName**.
- ➢ It uses two character variables to store the initials, **FirstInitial** and **LastInitial**.
- ➢ The trick to identifying which part of the input string is the first name and which part is the last name is to locate the blank space between them.
- ➢ Assume that the following arrays and variables have been declared:

  Character arrays: **FullName**[30], **FirstName**[15], **LastName**[15]

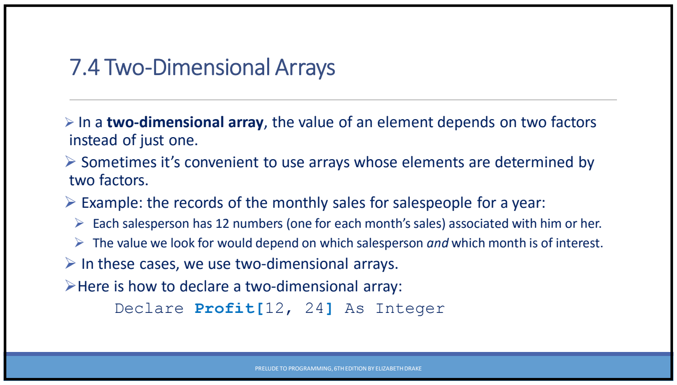
  Character variables: **FirstInitial**, **LastInitial**

  Integer variables: **J**, **K**, **Count**



## Example: Using **Character** Arrays (continued)

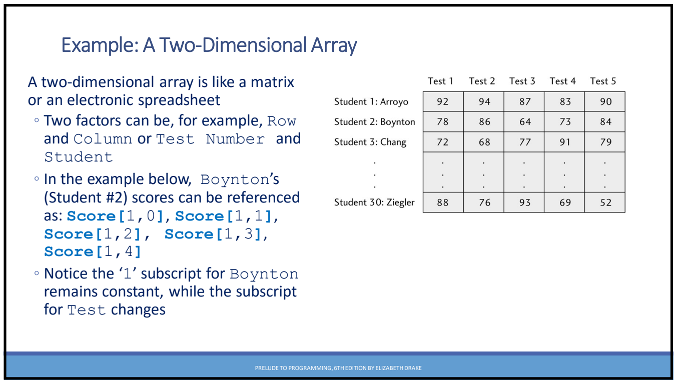
```
Write "Enter a name, first name first, then last name:"
Input FullName
Set Count = 0
While FullName[Count] != " "
       Set FirstName[Count] = FullName[Count]
       Set Count = Count + 1
End While
Set FirstInitial = FullName[0]
Set LastInitial = FullName[Count + 1]
Set J = 0
For (K = Count + 1; K <= Length_Of(FullName) – 1; K++)
       Set LastName[J] = FullName[K]
       Set J = J + 1
End For
Write LastName + ", " + FirstName
Write "Your initials are " + FirstInitial + LastInitial
```



## 7.4 Two-Dimensional Arrays

- In a **two-dimensional array**, the value of an element depends on two factors instead of just one.
- Sometimes it's convenient to use arrays whose elements are determined by two factors.
- Example: the records of the monthly sales for salespeople for a year:
  - Each salesperson has 12 numbers (one for each month's sales) associated with him or her.
  - The value we look for would depend on which salesperson *and* which month is of interest.
- In these cases, we use two-dimensional arrays.
- Here is how to declare a two-dimensional array:

```
Declare Profit[12, 24] As Integer
```

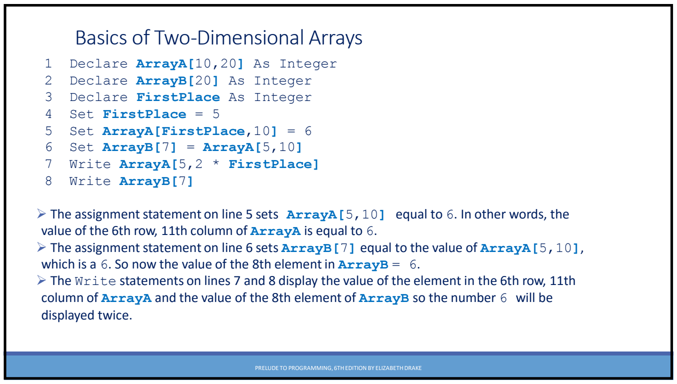


## Example: A Two-Dimensional Array

A two-dimensional array is like a matrix or an electronic spreadsheet

- Two factors can be, for example, `Row` and `Column` or `Test Number` and `Student`
- In the example below, `Boynton`'s (Student #2) scores can be referenced as: `Score[1,0]`, `Score[1,1]`, `Score[1,2]`, `Score[1,3]`, `Score[1,4]`
- Notice the '`1`' subscript for `Boynton` remains constant, while the subscript for `Test` changes

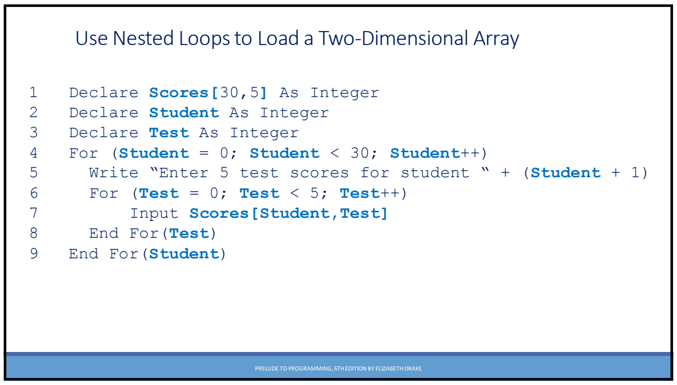
| | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 |
|---|---|---|---|---|---|
| Student 1: Arroyo | 92 | 94 | 87 | 83 | 90 |
| Student 2: Boynton | 78 | 86 | 64 | 73 | 84 |
| Student 3: Chang | 72 | 68 | 77 | 91 | 79 |
| .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. |
| Student 30: Ziegler | 88 | 76 | 93 | 69 | 52 |



## Basics of Two-Dimensional Arrays

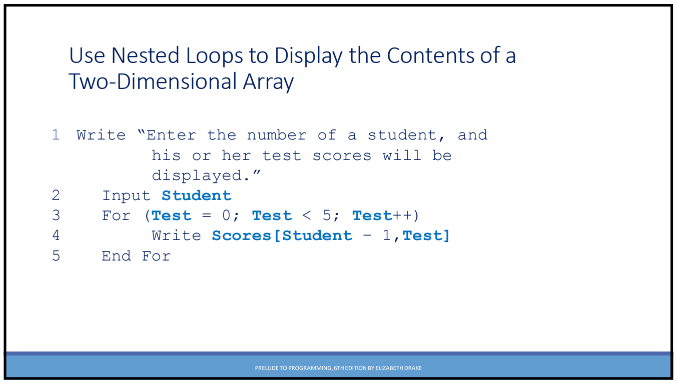
```
1  Declare ArrayA[10,20] As Integer
2  Declare ArrayB[20] As Integer
3  Declare FirstPlace As Integer
4  Set FirstPlace = 5
5  Set ArrayA[FirstPlace,10] = 6
6  Set ArrayB[7] = ArrayA[5,10]
7  Write ArrayA[5,2 * FirstPlace]
8  Write ArrayB[7]
```

- The assignment statement on line 5 sets `ArrayA[5,10]` equal to `6`. In other words, the value of the 6th row, 11th column of `ArrayA` is equal to `6`.
- The assignment statement on line 6 sets `ArrayB[7]` equal to the value of `ArrayA[5,10]`, which is a `6`. So now the value of the 8th element in `ArrayB = 6`.
- The `Write` statements on lines 7 and 8 display the value of the element in the 6th row, 11th column of `ArrayA` and the value of the 8th element of `ArrayB` so the number `6` will be displayed twice.



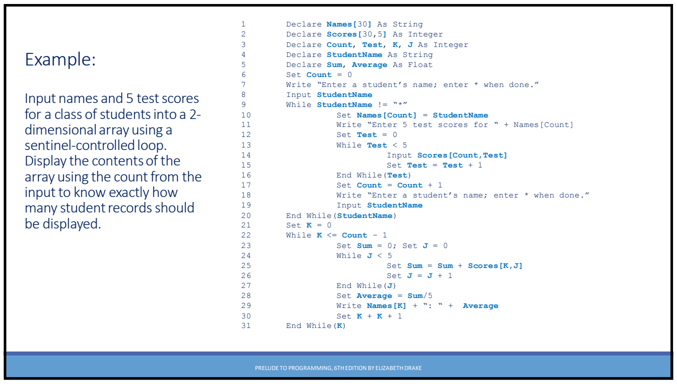
## Use Nested Loops to Load a Two-Dimensional Array

```
Declare Scores[30,5] As Integer
Declare Student As Integer
Declare Test As Integer
For (Student = 0; Student < 30; Student++)
  Write "Enter 5 test scores for student " + (Student + 1)
  For (Test = 0; Test < 5; Test++)
      Input Scores[Student,Test]
  End For(Test)
End For(Student)
```



## Use Nested Loops to Display the Contents of a Two-Dimensional Array

```
Write "Enter the number of a student, and
        his or her test scores will be
        displayed."
  Input Student
  For (Test = 0; Test < 5; Test++)
        Write Scores[Student - 1,Test]
  End For
```



## Example:

Input names and 5 test scores for a class of students into a 2-dimensional array using a sentinel-controlled loop. Display the contents of the array using the count from the input to know exactly how many student records should be displayed.

```
Declare Names[30] As String
Declare Scores[30,5] As Integer
Declare Count, Test, K, J As Integer
Declare StudentName As String
Declare Sum, Average As Float
Set Count = 0
Write "Enter a student's name; enter * when done."
Input StudentName
While StudentName != "*"
        Set Names[Count] = StudentName
        Write "Enter 5 test scores for " + Names[Count]
        Set Test = 0
        While Test < 5
                Input Scores[Count,Test]
                Set Test = Test + 1
        End While(Test)
        Set Count = Count + 1
        Write "Enter a student's name; enter * when done."
        Input StudentName
End While(StudentName)
Set K = 0
While K <= Count - 1
        Set Sum = 0; Set J = 0
        While J < 5
                Set Sum = Sum + Scores[K,J]
                Set J = J + 1
        End While(J)
        Set Average = Sum/5
        Write Names[K] + ": " +  Average
        Set K + K + 1
End While(K)
```