100 XP

Exercise - Write code to display all ourAnimals array data

15 minutes

In this exercise, you write the nested looping structure and selection code block that's used to display the ourAnimals array information. The detailed tasks that you complete during this exercise are:

1. Outer loop: build the outer loop that iterates through the animals in the ourAnimals array.
2. Data check: write code that checks for existing pet data and displays the pet ID if data exists for a pet.
3. Inner loop: build an inner loop that displays all of the pet characteristics for pets with data assigned.
4. Verification test: perform verification tests for the code that you develop in this exercise.

Important

You must complete the previous exercise in this module before starting this exercise.

Build a loop to iterate through the ourAnimals array

In this task, you create the outer `for` loop that's used to iterate through the animals in the `ourAnimals` array. You examine the relationship between the dimensions of your array and the parameters of your `for` loop. You also consider the differences between using `for` statements and `foreach` statements when working with multidimensional arrays.

1. Ensure that Visual Studio Code is open, and that your Program.cs file is visible in the Editor.

2. Inside the code block for the `switch(menuSelection)` selection statement, locate the following code lines:

   c#

```
case "1":
    // List all of our current pet information
    Console.WriteLine("this app feature is coming soon - please check back to
see progress.");
    Console.WriteLine("Press the Enter key to continue.");
    readResult = Console.ReadLine();
    break;
```

3. Delete the `Console.WriteLine()` statement used to display the "coming soon" message, and leave a blank code line below the `// List all of our current pet information` code comment line.

   The feedback message won't be needed because you'll be displaying the contents of the `ourAnimals` array. As you develop the code corresponding to the other menu selections, those feedback messages will be updated as well.

   Leave the message that says `Press the Enter key to continue.`

4. Starting on the blank code line that you created, begin a `for` statement as follows:

```c#
for ()
{
}
```

5. Take a minute to consider what you need to achieve with this `for` statement.

   Recall that a `for` statement has three parts that control how it implements iterations: the *for initializer*; the *for condition*, and the *for iterator*. The values assigned to the *for initializer*, *for condition*, and *for iterator* are based on what you need to achieve with the `for` statement.

   In this case, the `for` loop is used to iterate through the `ourAnimals` array. You know that arrays are zero indexed, meaning that an array with `n` elements is indexed from `0` to `n-1`. You need the *for initializer*, *for condition*, and *for iterator* to match dimensions of the array. In this case, you want the `for` loop to start at `0`, increment by `1`, and end at `maxPets-1`.

   Your `ourAnimals` array is declared as follows: `string[,] ourAnimals = new string[maxPets, 6];`. You know that the value assigned to `maxPets` is `8`. In this declaration, `maxPets` specifies the number of elements in the first dimension of the array, not the zero-based index number that you use to reference elements in the array. Therefore, although `maxPets = 8`, the array index numbers range from `0` to `7`.

6. To specify the control value of your `for` loop, update your code as follows:

   ```c#
   for (int i = 0; i < maxPets; i++)
   {
   }
   ```

   As you can see, setting the *for initializer* to `int i = 0;` aligns with the zero-based array index. Likewise, setting the *for condition* to `i < maxPets;` aligns with the first dimension of the array. Finally, setting the *for iterator* to `i++` will increment your loop control value by `1` for each iteration.

7. Take a minute to consider the choice between a `for` statement and a `foreach` statement when iterating through the ourAnimals array.

   The goal is to iterate through each animal in the ourAnimals array one at a time. So why not use a `foreach` loop? After all, you know that the `foreach` statement is designed for cases when you want to iterate through each item in an array of items.

   The reason why you don't use a `foreach` loop in this situation is because the `ourAnimals` array is multidimensional array. Since `ourAnimals` is a multidimensional string array, each element contained within `ourAnimals` is a separate item of type string. If you used a `foreach` loop to iterate through `ourAnimals`, the `foreach` would recognize each string as a separate item in a list of 48 string items (8 x 6 = 48). The `foreach` statement wouldn't process the two array dimensions separately. In other words, a `foreach` loop won't recognize `8` rows of string elements, where each row contains a column of `6` items. Since you want to work with one animal at a time, and process all six animal characteristics during a single iteration, a `foreach` statement isn't the right choice.

   However, if the `ourAnimals` array was a jagged array configured as an array of string arrays, you could use a `foreach` statement. In this case, you would create a `foreach` for an outer loop and second `foreach` for an inner loop. The outer loop would iterate through the "string array" elements in the jagged array. The string arrays are the "rows" in the two-dimensional array. The inner loop would iterate through the "string" elements contained in the string arrays. The string elements in the string arrays are the "columns" in the two-dimensional array.

The following code sample demonstrates the jagged array approach.

```c#
string[][] jaggedArray = new string[][]
{
    new string[] { "one1", "two1", "three1", "four1", "five1", "six1" },
    new string[] { "one2", "two2", "three2", "four2", "five2", "six2" },
    new string[] { "one3", "two3", "three3", "four3", "five3", "six3" },
    new string[] { "one4", "two4", "three4", "four4", "five4", "six4" },
    new string[] { "one5", "two5", "three5", "four5", "five5", "six5" },
    new string[] { "one6", "two6", "three6", "four6", "five6", "six6" },
    new string[] { "one7", "two7", "three7", "four7", "five7", "six7" },
    new string[] { "one8", "two8", "three8", "four8", "five8", "six8" }
};

foreach (string[] array in jaggedArray)
{
    foreach (string value in array)
    {
        Console.WriteLine(value);
    }
    Console.WriteLine();
}
```

For the Contoso Pets application, it's probably easier to use a multidimensional string array and nested `for` loops rather than a jagged array and nested `foreach` loops. Now that you see how each option works, you can make your own choice in future coding projects.

8. On the Visual Studio Code **File** menu, select **Save**.

9. Open the Integrated Terminal panel in Visual Studio Code and enter the command to Build your program.

   To open the Integrate Terminal from the EXPLORER view, right-click **Starter**, and then select **Open in Integrated Terminal**. You can also use the **View** or **Terminal** menu to open the Integrated Terminal panel.

   To Build your program, enter the `dotnet build` command from the Terminal command prompt.

10. Fix any build errors or warnings that you see reported before continuing.

Remember that the Build error and warning messages tell you what the issue is and where you can find it. When resolving issues, it's best to start with the issues that occur near the top of your code and work down.

11. Close the Terminal panel.

Check for existing pet data and display result

In this task, you use an `if` statement to find each pet in the `ourAnimals` array that has been assigned pet characteristics data. When a pet with assigned data is found, you display the petID. When there's no data assigned, nothing is displayed. You run the code to verify that your `for` and `if` statements are working correctly.

1. Create a blank code line inside your `for` statement code block as follows:

```c#
for (int i = 0; i < maxPets; i++)
{

}
```

2. To create an `if` statement that checks for pet ID data, update your code as follows:

```c#
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
    }
}
```

3. Take a minute to consider what this `if` statement is evaluating and why.

   First, consider the left side of the expression: `ourAnimals[i, 0]`. Notice that the loop control variable `i` is used to specify the animal that's being examined. As you may recall, the `0` in `[i, 0]` corresponds to the `petID` characteristic. Since the first dimension of the array corresponds to the "number" of the animal, this side of the expression ensures that your code checks the value assigned to `petID` for each animal in the array.

Second, consider the choice of comparison operator. Notice that the not-equal operator, `!=`, is being used. The expression evaluates as `true` whenever the value assigned to petID, `ourAnimals[i, 0]`, is NOT equal to the value listed on the right side of the equation.

Third, consider the value on the right side of the equation. Notice that a static string value of `"ID #: "` is used. This is the default value assigned to `petID` when the sample data is generated. When characteristics are assigned to an animal, the `petID` value is updated, and will NOT be equal to the default value.

This tells you that the code block of `if` statement will be executed when the current animal has characteristics defined.

> ⓘ **Note**
>
> This is a good example for when `!=` should be used. You don't care what value is assigned to `petID` as long as it's not the default value.

4. To create a `Console.WriteLine()` method that displays the `petID` inside the `if` statement's code block, update your code as follows:

```c#
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        Console.WriteLine(ourAnimals[i, 0]);
    }
}
```

Notice that an array element can be used as an argument when calling the `WriteLine()` method.

5. On the Visual Studio Code **File** menu, select **Save**.

6. Open the Integrated Terminal panel and enter the command to Build your program.

7. Fix any build errors or warnings that you see reported before continuing.

8. At the Terminal command prompt, enter the command to run your program.

Enter the `dotnet run` command at the Terminal command prompt to run your program code.

If your code generates a runtime error, fix the errors, save your updates, and restart the application.

9. At the Terminal command prompt, to verify that your new code logic is working as expected, enter **1**

   You should see the following `petID` values displayed:

   Output

   ```
   ID #: d1
   ID #: d2
   ID #: c3
   ID #: c4
   Press the Enter key to continue.
   ```

   These IDs correspond to the pets that have assigned data.

   If your code displays different output when you select menu option 1, review and update your code. Remember that you need to Save your Program.cs file after making updates.

10. Exit the application, and then close the Terminal panel.

Display all pet characteristics for pets with data assigned

In this task, you create a `for` loop inside the `if` statement code block that's used to display all of the characteristics of the current pet.

1. In the code Editor, locate the following code lines in your Program.cs file:

   c#

   ```csharp
   for (int i = 0; i < maxPets; i++)
   {
       if (ourAnimals[i, 0] != "ID #: ")
       {
           Console.WriteLine(ourAnimals[i, 0]);
   ```

```
    }
}
```

2. To create the `for` loop that will iterate through the characteristics of each pet, update your code as follows:

```c#
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        Console.WriteLine(ourAnimals[i, 0]);
        for (int j = 0; j < 6; j++)
        {
        }
    }
}
```

   Notice that you now have a second `for` loop that's "nested" inside the code block of the first `for` loop. As you know, the outer loop iterates through the animals in the `ourAnimals` array. The intention is for the inner loop to iterate through the characteristics of each animal. Since the animal data is stored in a multidimensional array, it will be easy to access animal characteristics.

3. Take a minute to review the `for` statement that you entered.

   Notice that the loop control variable is named `j`. When you nest `for` loops, one conventional approach is to use `i` in the outer loop and `j` in the inner loop. Following conventions like this makes it easier for others to read your code.

   Since six characteristics are stored for each animal, the *for initializer* is `int j = 0;` and the *for condition* is `j < 6;`. This combination of initializer and condition matches the array index range that you need, `0` - `5`.

4. To display each characteristic of a pet on a separate line, update your code as follows:

```c#
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        Console.WriteLine(ourAnimals[i, 0]);
        for (int j = 0; j < 6; j++)
        {
            Console.WriteLine(ourAnimals[i, j]);
        }
    }
}
```

5. Take a minute to consider the nested structure that you've created and the displayed output that your code will produce.

   Notice that the value written to the console, `ourAnimals[i, j]`, uses the loop control variables from both the outer and inner `for` loops.

   You know each of the following items:

   - The first dimension of the `ourAnimals` array corresponds to the different pets.
   - The second dimension of the `ourAnimals` array corresponds to the characteristics of each pet.
   - The `if` statement prevents the inner loop from running when there's no pet data assigned to the current pet.
   - The inner loop completes all of its iterations for each iteration of the outer loop.

   Therefore, you know that each animal's characteristics will be displayed as intended.

6. To replace the petID message with a blank `WriteLine()`, update your code as follows:

   ```c#
   for (int i = 0; i < maxPets; i++)
   {
       if (ourAnimals[i, 0] != "ID #: ")
       {
           Console.WriteLine();
           for (int j = 0; j < 6; j++)
           {
               Console.WriteLine(ourAnimals[i, j]);
           }
       }
   }
   ```

   This final update makes it easier to see the separation between pets when your output is displayed to the console.

7. On the Visual Studio Code **File** menu, select **Save**.

8. Open a Terminal pane and Build the program.

9. Fix any build errors or warnings that you see reported before continuing.

Check your work

In this task, you run your application from the Integrated Terminal and verify that your nested combination of `for` and `if` statements produces the expected result.

1. If necessary, open Visual Studio Code's Integrated Terminal panel.

2. At the Terminal command prompt, enter **dotnet run**

3. At the Terminal command prompt, enter **1**

4. Verify that the pet data is displayed for the four pets that have been assigned data.

Output

```
ID #: d1
Species: dog
Age: 2
Nickname: lola
Physical description: medium sized cream colored female golden retriever weigh-
ing about 65 pounds. housebroken.
Personality: loves to have her belly rubbed and likes to chase her tail. gives
lots of kisses.

ID #: d2
Species: dog
Age: 9
Nickname: loki
Physical description: large reddish-brown male golden retriever weighing about
85 pounds. housebroken.
Personality: loves to have his ears rubbed when he greets you at the door, or at
any time! loves to lean-in and give doggy hugs.

ID #: c3
Species: cat
Age: 1
Nickname: Puss
Physical description: small white female weighing about 8 pounds. litter box
trained.
Personality: friendly
```

```
ID #: c4
Species: cat
Age: ?
Nickname:
Physical description:
Personality:
Press the Enter key to continue.
```

Note

If you don't see the expected results displayed, ensure that you saved your updated Program.cs file. If you're not seeing the expected results and you can't identify the issue, you can examine the Program.cs code in the Final folder. The Final folder is included as part of the download you completed during Setup. We recommend that you spend time trying to identify and fix syntax and logic issue in your code before checking the Program.cs file in the Final folder.

5. Exit the application, and then close the Terminal panel.

Next unit: Exercise - Build and test a loop for entering new pet data

Continue >