CSC107

Collections

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

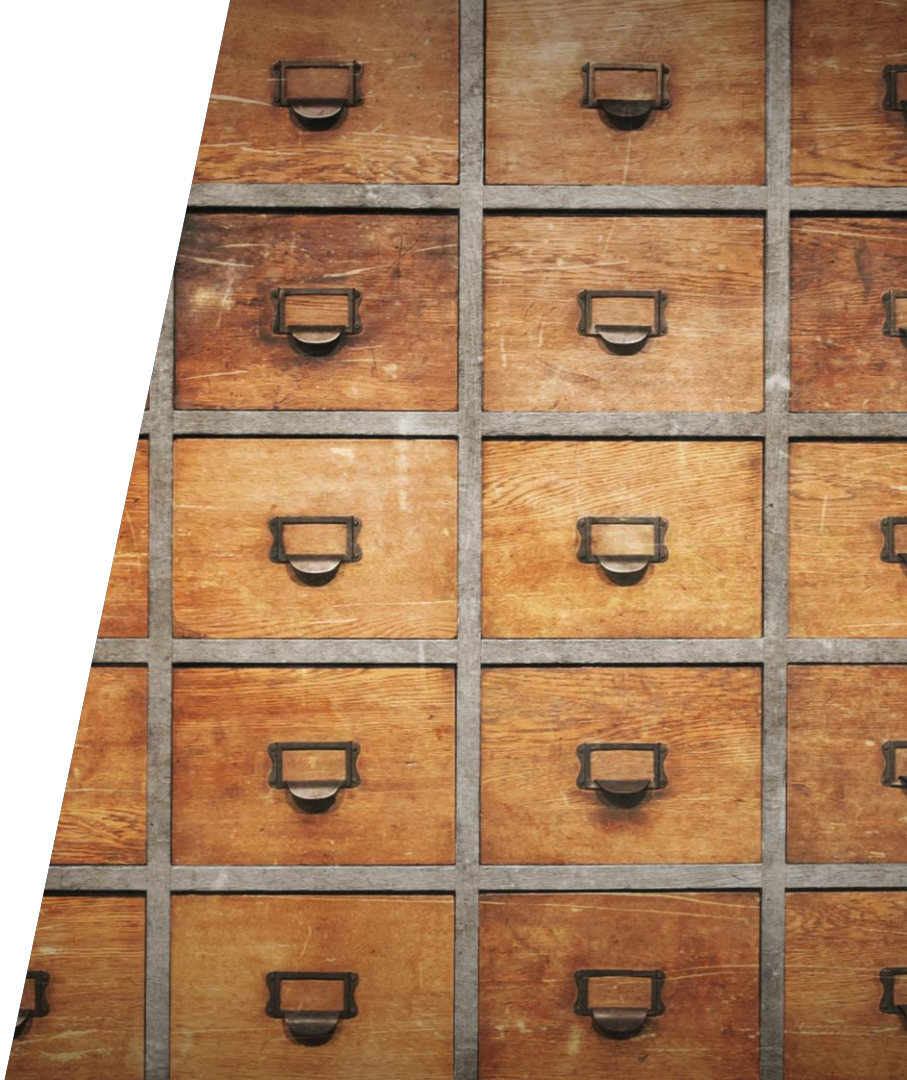
1. Use arrays to load static data sets
2. Use lists and dictionaries to manage data

Use arrays to load static data sets

Xamarin University

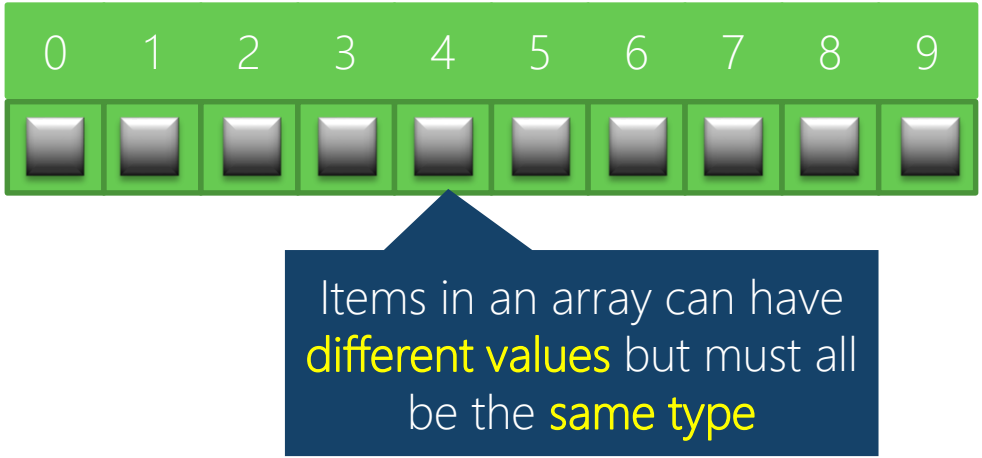
Tasks

1. Declare arrays
2. Assign arrays
3. Iterate over data in arrays



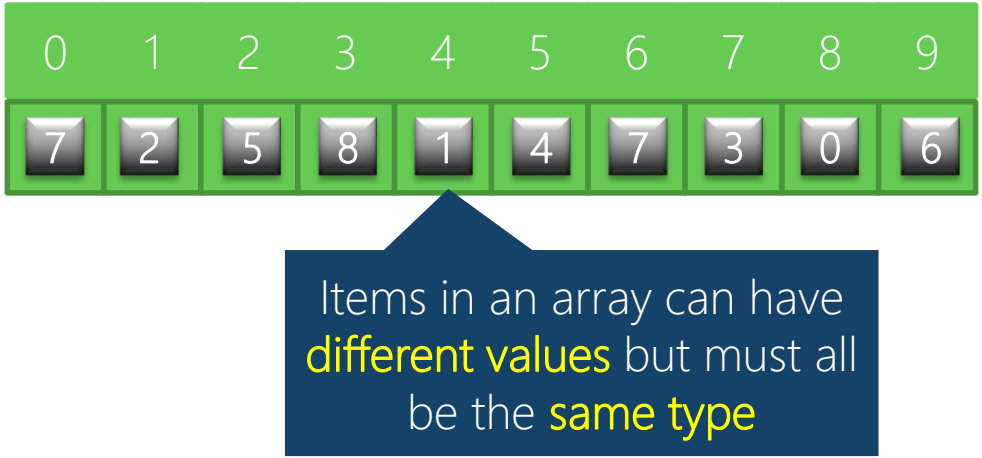
What are Arrays?

- Arrays are ordered collections of homogenous (same type) data



What are Arrays?

- Arrays are ordered collections of homogenous (same type) data



How do I create an array?

- When created, arrays must have a **type**, a **name** (identifier) and must be told the **number of elements** the array will hold

identifier

number of elements

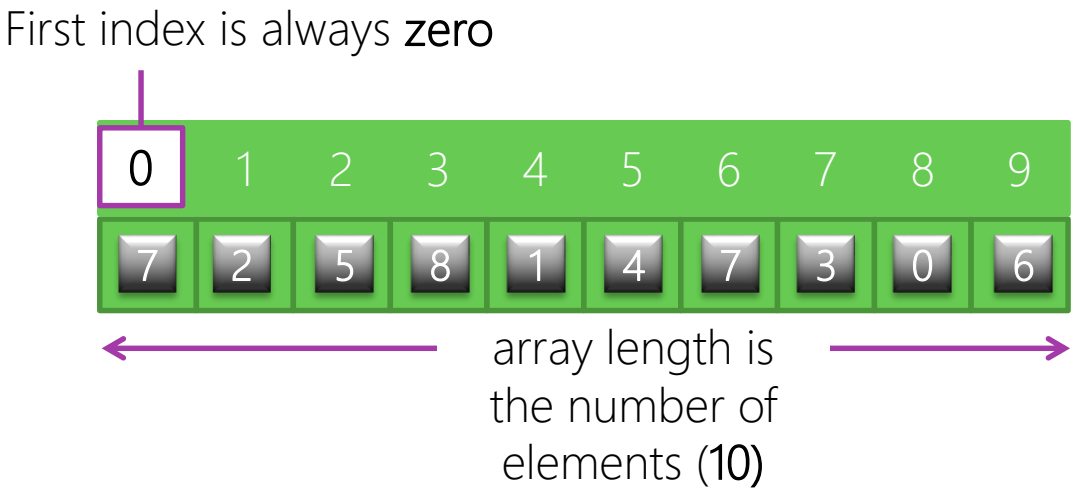
```
int[] myFirstArray = new int[10];
```

To declare the field for an array, use brackets `[]` after the type

type of the elements in the array

How arrays are indexed

- Each item has an **index**, which is an offset from the start



Setting item values

- Arrays are initialized to empty values (typically zero), but you can set values using the index position

```
int[] numbers = new int[10];

numbers[0] = 3;
numbers[1] = 1;
numbers[2] = 4;
numbers[3] = 1;
numbers[4] = 5;
...
```

This array holds integers

C# will ensure you only put integers in this array – other values will cause a compile error

Getting item values

- Use the same indexer syntax to *retrieve* values

```
int[] numbers = new int[10];
...

int firstValue = numbers[0];
int fourthValue = numbers[3];
```

C# will also ensure you retrieve the correct type from the array at compile time

Getting item values

- Use the same indexer syntax to *retrieve* values

```
int[] numbers = new int[10];
...

int firstValue = numbers[0];
int fourthValue = numbers[3];

int badValue = numbers[10];
```

Crashes program
Why?

How can I tell how many items there are?

- You can find out how many elements are in an array by using the **`Length`** property

There are **5 elements**, indexed 0 – 4.

```
int[] arrayA = new int[5];
int lengthA = arrayA.Length;
Console.WriteLine(lengthA); // Writes 5
```

Flash Quiz

Xamarin University

Flash Quiz

① What does the **type** of an array tell you?

a) The type of the array itself

b) The type of the objects in the array

Flash Quiz

① What does the type of an array tell you?

a) The type of the array itself

b) The type of the objects in the array

Flash Quiz

② How do I access the 5th element in an array named **theArray**?

a) theArray[5]

b) theArray[4]

c) theArray[6]

Flash Quiz

② How do I access the 5th element in an array named **theArray**?

a) theArray[5]

b) theArray[4]

c) theArray[6]

Flash Quiz

③ If I create an array: `int[] myArray = new int[5];`

and then write: `int x = myArray[5];`

what will happen?

a) x will be assigned the value 5

b) the program will crash when you build

c) the program will crash when it runs

Flash Quiz

③ If I create an array: `int[] myArray = new int[5];`

and then write: `int x = myArray[5];`

what will happen?

a) x will be assigned the value 5

b) the program will crash when you build

c) <u>the program will crash when it runs</u>

Arrays as parameters

- Arrays can be passed as parameters which allows the program to store and modify multiple items without managing multiple variables

```
int[] numbers = new int[10];
int sum = CalculateSum(numbers);
```

```
int CalculateSum (int[] numbersToSum)
{
    int total = 0;
    ...
    return total;
}
```

Pass all 10 numbers together to this function

Working with the array data

- You can **iterate** through an array with the **for** statement – this allows you to process each item by the index in a loop

Notice < **length** to stay within bounds

```
int[] numbers = new int[20];

for (int counter = 0; counter < numbers.Length; counter++) {
    numbers[counter] = counter * 2;
}
```

This approach is very flexible because we define the range to process – e.g. where to start and end

Fun Fact

- Many programmers use **i** and **j** as counters in for loops

```
for ( int i = 0; i < total; i++) {
    // Code is here
}
```

- The reason is that a very old programming language (Fortran) reserved **i**, **j**, **k** and **l** as the counter variable names, and old habits die hard.

Working with the array data

- Alternatively, C# provides the **foreach** statement which allows you to process each item without a specific loop counter or length

```
int[] numbers = new int[20];

foreach ( int current in numbers ) {
    Console.Write( current.ToString() + " " );
}
```

foreach does not supply an index, instead there is **only a value**, so you cannot change the array contents with this approach

Individual Exercise

Creating and displaying arrays

Xamarin University

Arrays of user-defined objects

- Items in arrays can be **user-defined objects**

An array of 10 Dogs

```
Dog[] dogs = new Dog[10];
Person[] crowd = new Person[100];
Employee[] business = new Employee[250];
```

An array of 250 Employees

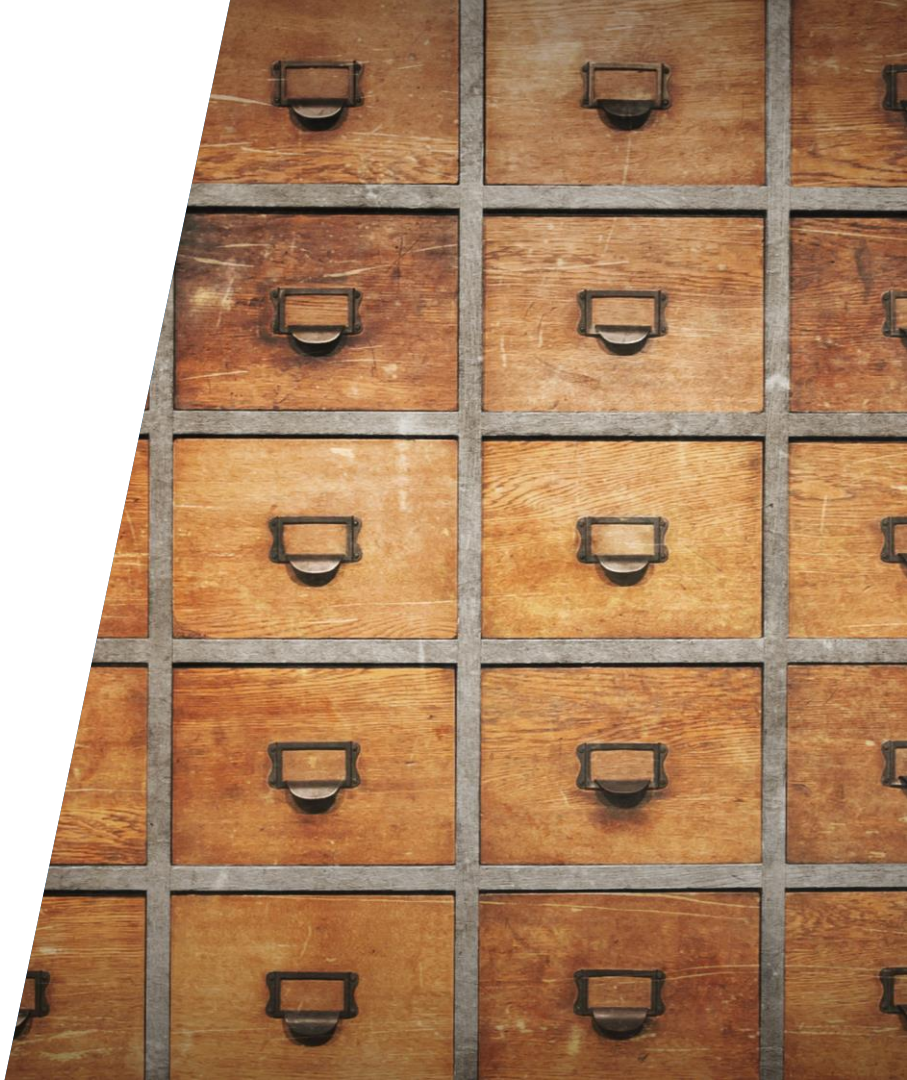
Individual Exercise

Create array of Employees

Xamarin University

Summary

1. Declare arrays
2. Assign arrays
3. Iterate over data in arrays



Use lists and dictionaries to manage data

Xamarin University

Tasks

1. Discuss limitations of arrays
2. Introduce Lists
3. Use dictionaries to manage data

What's wrong with arrays?

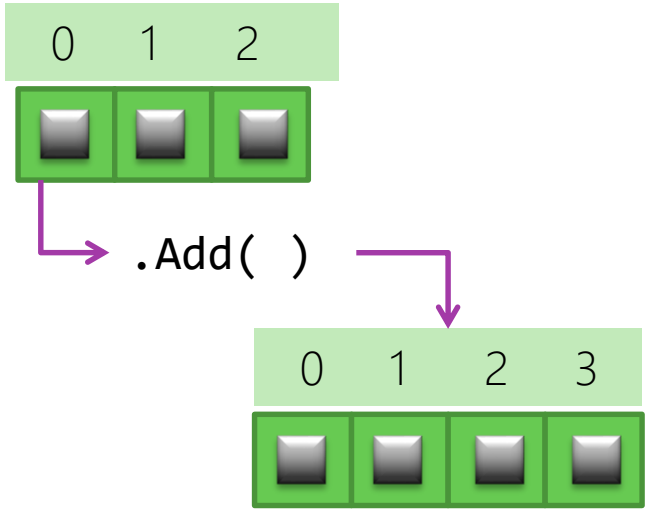
- Arrays are **fixed in size**, must decide at the start how many elements you will have – **what if you don't know**?

```
public class Facebook
{
    string[] friends = new string[10];
    ...
}
```

How many friends should we allow? 10? 100? 1000?

Introducing: Lists

- Lists are **dynamic** and can **grow** and **contract** as necessary, but act like arrays in most other ways
  - Can use `foreach` to iterate through the items
  - Can get or set values using index positions (`[0]`)
  - Holds heterogeneous values of a defined type



Creating Lists

- Declaring a **`List`** is similar to arrays, but the type is in angle brackets **`<>`**

identifier

```
List<int> myFirstList = new List<int>();
```

Type of objects in list

do **not** pass defined length

Adding values to a list

- Can append new items to the list with the **Add** method, the object you add must match the type of the list

Add value 15 to the list

```
List<int> myFirstList = new List<int>();
myFirstList.Add(15);
myFirstList.Add(20);


...
```

Adding values to a list

- Can append new items to the list with the **Add** method, the object you add must match the type of the list

```
List<int> myFirstList = new List<int>();
myFirstList.Add(15);
myFirstList.Add(20);

List<int> mySecondList = new List<int>();
for (int i = 0; i < 100; i++) {
    mySecondList.Add(i * 2);
}
```

Add 100 values to the second list

What is a generic type?

- List is called a *generic* because it can be created to hold any type of object, specified in the angle brackets
- In some books you'll see List specified as `List<T>`
  - pronounced as "List of T" where T stands for type (you can use any letter)
  - You substitute the actual type for `T`, for example `List<Dog>`

```
List<int> numbers = ...;

List<string> names = ...;

List<Dog> dogs = ...;

List<Person> people = ...;
```

Lists can be defined to hold any type you need, including user-defined types

Iterate over a List

- You typically process the items in a list with **foreach**

```
List<Dog> kennel = List<Dog>();
GetAllOurDoggieGuests(kennel);
...
foreach (Dog thisDog in kennel) {
  thisDog.Bark();
}
```

Loop through each **Dog** in the list, ask each one to **Bark**

Flash Quiz

Xamarin University

Flash Quiz

① How do I set the size in a list?

a) At the time you create it

b) At the time you use it

c) You don't

Flash Quiz

① How do I set the size in a list?

a) At the time you create it

b) At the time you use it

c) <u>You don't</u>

Flash Quiz

② How do you declare a list of Employees?

a) Employee employees = new List();

b) Employee employees = new List(Employee);

c) List<Employee> employees = new List;

d) List<Employee> employees = new List<Employee>;

e) List<Employee> employees = new List<Employee>();

Flash Quiz

② How do you declare a list of Employees?

a) Employee employees = new List();

b) Employee employees = new List(Employee);

c) List<Employee> employees = new List;

d) List<Employee> employees = new List<Employee>;

e) List<Employee> employees = new List<Employee>();

Individual Exercise

Create a List of Employees

Xamarin University

What are Dictionaries?

- Dictionaries are collections that store values a key – instead of indexes, all operations in the collection are performed with keys
  - Keys must be unique
  - Keys can be any type but must be consistent in the dictionary
  - Values can be any type but must be consistent in the dictionary

Dictionaries

- You define a dictionary just as you define a **`List`**, but you specify both the **type of the key** and of the **value**

```
var employees = new Dictionary<string, Employee>();
```

`var` tells C# to define the field as whatever is being assigned – in this case the dictionary, so we don't have to type it twice!

Type of key

Type of value

Adding values to a dictionary

- Can add values to the dictionary using the **Add** method

```
var employees = new Dictionary<string, Employee>();

employees.Add( "Jesse", new Employee() );
employees.Add( "Mark", new Employee() );
employees.Add( "Zulma", new Employee() );
...
```

Adding values to a dictionary

- Can **add or replace** values to the dictionary using the key as an index

```
var employees = new Dictionary<string, Employee>();

employees.Add( "Jesse", new Employee() );
...
employees["Jesse"] = new Employee();  // replace
employees["Moe"] = new Employee();    // add new
```

Using keys to access values

- You access values in a dictionary by using the key as an "index", the **key must exist** or a runtime error occurs

```
var employees = new Dictionary<string, Employee>();
employees.Add( "Jesse", new Employee() );

Employee employeeOfTheMonth = employees["Jesse"];
```

holds `Employee` object (value) at key (`"Jesse"`)

dictionary

key

Checking for a key

- Can use **`TryGetValue`** to test for a key when it might not exist

```
var employees = new Dictionary<string, Employee>();
employees.Add( "Jesse", new Employee() );

Employee employeeOfTheMonth;
if (employees.TryGetValue("Jesse", out employeeOfTheMonth)) {
    // Use employeeOfTheMonth
}
else {
    // Not found
}
```

Return value is a bool

Value we care about returned as an out parameter

Individual Exercise

Create a dictionary of Employees keyed by EmployeeID

Xamarin University

Summary

1. Discuss limitations of arrays
2. Introduce Lists
3. Use dictionaries to manage data

Where are we going from here?

- You now know how to handle collections of data with arrays and lists
- In the next course, we will look at how to deal with runtime failures in our applications by *handling exceptions*

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft