LISTS AND ARRAYS

Topics

Topics

- C# Collections

Topics

- C# Collections
- List

Topics

- C# Collections
- List
  - Flexible collection of variable length

Topics

- C# Collections
- List
  - Flexible collection of variable length
- Array

Topics

- C# Collections
- List
  - Flexible collection of variable length
- Array
  - Standard arrays

Topics

- C# Collections
- List
  - Flexible collection of variable length
- Array
  - Standard arrays
- Multidimensional Arrays

Topics

- C# Collections
- List
  - Flexible collection of variable length
- Array
  - Standard arrays
- Multidimensional Arrays
- Jagged Lists & Arrays

Topics

- C# Collections
- List
  - Flexible collection of variable length
- Array
  - Standard arrays
- Multidimensional Arrays
- Jagged Lists & Arrays
- `foreach` and Collections

Topics

- C# Collections
- List
  - Flexible collection of variable length
- Array
  - Standard arrays
- Multidimensional Arrays
- Jagged Lists & Arrays
- `foreach` and Collections
- When to Use List or Array

Topics

- C# Collections
- List
  - Flexible collection of variable length
- Array
  - Standard arrays
- Multidimensional Arrays
- Jagged Lists & Arrays
- `foreach` and Collections
- When to Use List or Array
- Other Collection Types

C# Collections

C# Collections

- A collection in C# is a group of several things that are referenced by a single variable

C# Collections

- A collection in C# is a group of several things that are referenced by a single variable
- Similar to a pride of lions, parliament of rooks, murder of crows

C# Collections

- A collection in C# is a group of several things that are referenced by a single variable
- Similar to a pride of lions, parliament of rooks, murder of crows

Image from http://www.quickmeme.com/meme/35hla2

C# Collections

- A collection in C# is a group of several things that are referenced by a single variable
- Similar to a pride of lions, parliament of rooks, murder of crows
- Two most important C# collections are:

C# Collections

- A collection in C# is a group of several things that are referenced by a single variable
- Similar to a pride of lions, parliament of rooks, murder of crows
- Two most important C# collections are:
  - List

C# Collections

- A collection in C# is a group of several things that are referenced by a single variable
- Similar to a pride of lions, parliament of rooks, murder of crows
- Two most important C# collections are:
  - List
  - Array

C# Collections

- A collection in C# is a group of several things that are referenced by a single variable
- Similar to a pride of lions, parliament of rooks, murder of crows
- Two most important C# collections are:
  - List
  - Array
- Both Lists and arrays do appear in the Inspector

C# Collections

- A collection in C# is a group of several things that are referenced by a single variable
- Similar to a pride of lions, parliament of rooks, murder of crows
- Two most important C# collections are:
  - List
  - Array
- Both Lists and arrays do appear in the Inspector
- List is the most flexible and easiest to use, so we'll start with List

List

List

- Requires a new `using` line at the top of your script

List

- Requires a new `using` line at the top of your script

```
using System.Collections.Generic;
```

List

- Requires a new `using` line at the top of your script

```
using System.Collections.Generic;
```

- List is a *generic collection*

List

- Requires a new `using` line at the top of your script

```
using System.Collections.Generic;
```

- List is a *generic collection*
  - Generic collections can work for any data type

List

- Requires a new `using` line at the top of your script

```
using System.Collections.Generic;
```

- List is a *generic collection*
  - Generic collections can work for any data type

```
List<string> sList;          // A List of strings
List<GameObject> goList;     // A List of GameObjects
```

List

- Requires a new `using` line at the top of your script

```
using System.Collections.Generic;
```

- List is a *generic collection*
  - Generic collections can work for any data type

```
List<string> sList;          // A List of strings
List<GameObject> goList;     // A List of GameObjects
```

- A List must be *defined* before it can be used (because Lists default to `null`)

List

- Requires a new `using` line at the top of your script

```
using System.Collections.Generic;
```

- List is a *generic collection*
  - Generic collections can work for any data type

```
List<string> sList;          // A List of strings
List<GameObject> goList;     // A List of GameObjects
```

- A List must be *defined* before it can be used (because Lists default to `null`)

```
sList = new List<string>();
```

List

- Requires a new `using` line at the top of your script

```
using System.Collections.Generic;
```

- List is a *generic collection*
  - Generic collections can work for any data type

```
List<string> sList;          // A List of strings
List<GameObject> goList;     // A List of GameObjects
```

- A List must be *defined* before it can be used (because Lists default to `null`)

```
sList = new List<string>();
```

- Elements are added to Lists using the Add() method

List

- Requires a new `using` line at the top of your script

```
using System.Collections.Generic;
```

- List is a *generic collection*
  - Generic collections can work for any data type

```
List<string> sList;          // A List of strings
List<GameObject> goList;     // A List of GameObjects
```

- A List must be *defined* before it can be used (because Lists default to `null`)

```
sList = new List<string>();
```

- Elements are added to Lists using the Add() method

```
sList.Add("Hello");
sList.Add("World");
```

List

List

- List elements are accessed via *bracket access*

List

- List elements are accessed via *bracket access*
  - Bracket access is *zero indexed* (i.e., starts at `[0]`)

List

- List elements are accessed via *bracket access*
  - Bracket access is *zero indexed* (i.e., starts at `[0]`)

```
print( sList[0] );          // Prints: "Hello"
print( sList[1] );          // Prints: "World"
```

List

- List elements are accessed via *bracket access*
  - Bracket access is *zero indexed* (i.e., starts at `[0]`)

```
print( sList[0] );          // Prints: "Hello"
print( sList[1] );          // Prints: "World"
```

- Lists have a Count of the number of elements

List

- List elements are accessed via *bracket access*
  - Bracket access is *zero indexed* (i.e., starts at `[0]`)

```
print( sList[0] );          // Prints: "Hello"
print( sList[1] );          // Prints: "World"
```

- Lists have a Count of the number of elements

```
print( sList.Count );       // Prints: "2"
```

List

- List elements are accessed via *bracket access*
  - Bracket access is *zero indexed* (i.e., starts at `[0]`)

```
print( sList[0] );          // Prints: "Hello"
print( sList[1] );          // Prints: "World"
```

- Lists have a Count of the number of elements

```
print( sList.Count );       // Prints: "2"
```

- Lists can be cleared of all elements

List

- List elements are accessed via *bracket access*
  - Bracket access is *zero indexed* (i.e., starts at `[0]`)

```
print( sList[0] );          // Prints: "Hello"
print( sList[1] );          // Prints: "World"
```

- Lists have a Count of the number of elements

```
print( sList.Count );       // Prints: "2"
```

- Lists can be cleared of all elements

```
sList.Clear();              // Empties sList
```

List

List

- All these methods act on the List ["A","B","C","D"]

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );                    // Prints: "A"
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
sList.IndexOf("B");         // 1   ("B" is the 1st element)
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
sList.IndexOf("B");         // 1   ("B" is the 1st element)
sList.IndexOf("Bob");       // -1   ("Bob" is not in the List)
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
sList.IndexOf("B");         // 1   ("B" is the 1st element)
sList.IndexOf("Bob");       // -1   ("Bob" is not in the List)
sList.Insert(2,"X");        // ["A","B","X","C","D"]
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
sList.IndexOf("B");         // 1   ("B" is the 1st element)
sList.IndexOf("Bob");       // -1   ("Bob" is not in the List)
sList.Insert(2,"X");        // ["A","B","X","C","D"]
sList.Insert(4,"X");        // ["A","B","C","D","X"]
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
sList.IndexOf("B");         // 1   ("B" is the 1st element)
sList.IndexOf("Bob");       // -1   ("Bob" is not in the List)
sList.Insert(2,"X");        // ["A","B","X","C","D"]
sList.Insert(4,"X");        // ["A","B","C","D","X"]
sList.Insert(5,"X");        // ERROR!!!   Index out of range
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
sList.IndexOf("B");         // 1   ("B" is the 1st element)
sList.IndexOf("Bob");       // -1   ("Bob" is not in the List)
sList.Insert(2,"X");        // ["A","B","X","C","D"]
sList.Insert(4,"X");        // ["A","B","C","D","X"]
sList.Insert(5,"X");        // ERROR!!!   Index out of range
sList.Remove("C");          // ["A","B","D"]
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
sList.IndexOf("B");         // 1   ("B" is the 1st element)
sList.IndexOf("Bob");       // -1   ("Bob" is not in the List)
sList.Insert(2,"X");        // ["A","B","X","C","D"]
sList.Insert(4,"X");        // ["A","B","C","D","X"]
sList.Insert(5,"X");        // ERROR!!!   Index out of range
sList.Remove("C");          // ["A","B","D"]
sList.RemoveAt(1);          // ["A","C","D"]
```

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
sList.IndexOf("B");         // 1   ("B" is the 1st element)
sList.IndexOf("Bob");       // -1   ("Bob" is not in the List)
sList.Insert(2,"X");        // ["A","B","X","C","D"]
sList.Insert(4,"X");        // ["A","B","C","D","X"]
sList.Insert(5,"X");        // ERROR!!!   Index out of range
sList.Remove("C");          // ["A","B","D"]
sList.RemoveAt(1);          // ["A","C","D"]
```

- Lists can be converted to arrays

List

- All these methods act on the List ["A","B","C","D"]

```
print( sList[0] );          // Prints: "A"
sList.Add("Apple");         // ["A","B","C","D","Apple"]
sList.Clear();              // []
sList.IndexOf("B");         // 1   ("B" is the 1st element)
sList.IndexOf("Bob");       // -1   ("Bob" is not in the List)
sList.Insert(2,"X");        // ["A","B","X","C","D"]
sList.Insert(4,"X");        // ["A","B","C","D","X"]
sList.Insert(5,"X");        // ERROR!!!   Index out of range
sList.Remove("C");          // ["A","B","D"]
sList.RemoveAt(1);          // ["A","C","D"]
```

- Lists can be converted to arrays

```
string[] sArray = sList.ToArray();
```

Array

Array

- Array is a much simpler collection than List

Array

- **Array is a much simpler collection than List**
  - **Does not require any `using` statement at the top of a script**

Array

- Array is a much simpler collection than List
  - Does not require any `using` statement at the top of a script
- Not actually its own data type, but rather a collection of any data type

Array

- Array is a much simpler collection than List
  - Does not require any `using` statement at the top of a script
- Not actually its own data type, but rather a collection of any data type

```
string[] sArray;                // An array of strings
GameObject[] goArray;           // An array of GameObjects
```

Array

- Array is a much simpler collection than List
  - Does not require any `using` statement at the top of a script
- Not actually its own data type, but rather a collection of any data type

```
string[] sArray;                // An array of strings
GameObject[] goArray;           // An array of GameObjects
```

- Arrays are created with a fixed length

Array

- Array is a much simpler collection than List
  - Does not require any `using` statement at the top of a script
- Not actually its own data type, but rather a collection of any data type

```
string[] sArray;                // An array of strings
GameObject[] goArray;           // An array of GameObjects
```

- Arrays are created with a fixed length
  - Arrays cannot expand to add more elements like Lists can

Array

- Array is a much simpler collection than List
  - Does not require any `using` statement at the top of a script
- Not actually its own data type, but rather a collection of any data type

```
string[] sArray;              // An array of strings
GameObject[] goArray;         // An array of GameObjects
```

- Arrays are created with a fixed length
  - Arrays cannot expand to add more elements like Lists can

```
sArray = new string[4];       // An array of four strings
```

Array

- Array is a much simpler collection than List
  - Does not require any `using` statement at the top of a script
- Not actually its own data type, but rather a collection of any data type

```
string[] sArray;              // An array of strings
GameObject[] goArray;         // An array of GameObjects
```

- Arrays are created with a fixed length
  - Arrays cannot expand to add more elements like Lists can

```
sArray = new string[4];       // An array of four strings
sArray = new string[] {"A","B","C","D"};
```

Array

- Array is a much simpler collection than List
  - Does not require any `using` statement at the top of a script
- Not actually its own data type, but rather a collection of any data type

```
string[] sArray;              // An array of strings
GameObject[] goArray;         // An array of GameObjects
```

- Arrays are created with a fixed length
  - Arrays cannot expand to add more elements like Lists can

```
sArray = new string[4];       // An array of four strings
sArray = new string[] {"A","B","C","D"};
```

  - Either of these arrays will only ever have a Length of 4

Array

Array

- Array elements are both accessed and assigned via *bracket access*

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
```

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
print( sArray[1] );       // Prints: "Bob"
```

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
print( sArray[1] );       // Prints: "Bob"
```

- It's possible to skip elements in an array

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
print( sArray[1] );       // Prints: "Bob"
```

- It's possible to skip elements in an array
  - The skipped elements are the default value for that type

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
print( sArray[1] );       // Prints: "Bob"
```

- It's possible to skip elements in an array
  - The skipped elements are the default value for that type

```
string[] sArray = new string[4];      // 4-element string array
```

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
print( sArray[1] );       // Prints: "Bob"
```

- It's possible to skip elements in an array
  - The skipped elements are the default value for that type

```
string[] sArray = new string[4];      // 4-element string array
sArray[0] = "A";                      // ["A",null,null,null]
```

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
print( sArray[1] );       // Prints: "Bob"
```

- It's possible to skip elements in an array
  - The skipped elements are the default value for that type

```
string[] sArray = new string[4];      // 4-element string array
sArray[0] = "A";                      // ["A",null,null,null]
sArray[2] = "C";                      // ["A",null,"C",null]
```

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
print( sArray[1] );       // Prints: "Bob"
```

- It's possible to skip elements in an array
  - The skipped elements are the default value for that type

```
string[] sArray = new string[4];      // 4-element string array
sArray[0] = "A";                      // ["A",null,null,null]
sArray[2] = "C";                      // ["A",null,"C",null]
```

- Arrays have Length instead of Count

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
print( sArray[1] );       // Prints: "Bob"
```

- It's possible to skip elements in an array
  - The skipped elements are the default value for that type

```
string[] sArray = new string[4];      // 4-element string array
sArray[0] = "A";                      // ["A",null,null,null]
sArray[2] = "C";                      // ["A",null,"C",null]
```

- Arrays have Length instead of Count

```
print( sArray.Length );               // Prints: 4
```

Array

- Array elements are both accessed and assigned via *bracket access*

```
sArray[1] = "Bob";        // Assigns "Bob" to the 1st element
print( sArray[1] );       // Prints: "Bob"
```

- It's possible to skip elements in an array
  - The skipped elements are the default value for that type

```
string[] sArray = new string[4];      // 4-element string array
sArray[0] = "A";                      // ["A",null,null,null]
sArray[2] = "C";                      // ["A",null,"C",null]
```

- Arrays have Length instead of Count

```
print( sArray.Length );               // Prints: 4
```

  - This is similar to strings (which are collections of chars)

Array

Array

- All these methods act on the array ["A","B","C","D"]

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );                    // Prints: "A"
```

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );          // Prints: "A"
sArray[2] = "Cow";           // ["A","B","Cow","D"]
```

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );          // Prints: "A"
sArray[2] = "Cow";           // ["A","B","Cow","D"]
sArray[10] = "Test";         // ERROR!!!  Index out of range
```

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );          // Prints: "A"
sArray[2] = "Cow";           // ["A","B","Cow","D"]
sArray[10] = "Test";         // ERROR!!!  Index out of range
sList.Remove("C");           // ["A","B","D"]
```

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );          // Prints: "A"
sArray[2] = "Cow";           // ["A","B","Cow","D"]
sArray[10] = "Test";         // ERROR!!!  Index out of range
sList.Remove("C");           // ["A","B","D"]
sList.RemoveAt(1);           // ["A","C","D"]
```

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );          // Prints: "A"
sArray[2] = "Cow";           // ["A","B","Cow","D"]
sArray[10] = "Test";         // ERROR!!!  Index out of range
sList.Remove("C");           // ["A","B","D"]
sList.RemoveAt(1);           // ["A","C","D"]
```

- Static methods of the System.Array class

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );          // Prints: "A"
sArray[2] = "Cow";           // ["A","B","Cow","D"]
sArray[10] = "Test";         // ERROR!!!  Index out of range
sList.Remove("C");           // ["A","B","D"]
sList.RemoveAt(1);           // ["A","C","D"]
```

- Static methods of the System.Array class

```
print( System.Array.IndexOf(sArray,"B") ); // 1
```

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );          // Prints: "A"
sArray[2] = "Cow";           // ["A","B","Cow","D"]
sArray[10] = "Test";         // ERROR!!!  Index out of range
sList.Remove("C");           // ["A","B","D"]
sList.RemoveAt(1);           // ["A","C","D"]
```

- Static methods of the System.Array class

```
print( System.Array.IndexOf(sArray,"B") ); // 1
System.Array.Resize( ref sArray, 6);     // Sets Length to 6
```

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );          // Prints: "A"
sArray[2] = "Cow";           // ["A","B","Cow","D"]
sArray[10] = "Test";         // ERROR!!!  Index out of range
sList.Remove("C");           // ["A","B","D"]
sList.RemoveAt(1);           // ["A","C","D"]
```

- Static methods of the System.Array class

```
print( System.Array.IndexOf(sArray,"B") ); // 1
System.Array.Resize( ref sArray, 6);       // Sets Length to 6
```

- Arrays can be converted to Lists

Array

- All these methods act on the array ["A","B","C","D"]

```
print( sArray[0] );          // Prints: "A"
sArray[2] = "Cow";           // ["A","B","Cow","D"]
sArray[10] = "Test";         // ERROR!!!  Index out of range
sList.Remove("C");           // ["A","B","D"]
sList.RemoveAt(1);           // ["A","C","D"]
```

- Static methods of the System.Array class

```
print( System.Array.IndexOf(sArray,"B") ); // 1
System.Array.Resize( ref sArray, 6);     // Sets Length to 6
```

- Arrays can be converted to Lists

```
List<string> sList = new List<string>( sArray );
```

Multidimensional Arrays

Multidimensional Arrays

- Arrays can have more than one dimension

Multidimensional Arrays

- Arrays can have more than one dimension

```
string[,] s2D = new string[4,4];      // Makes a 4x4 array
s2D[0,0] = "A";
s2D[0,3] = "B";
s2D[1,2] = "C";
s2D[3,1] = "D";
```

Multidimensional Arrays

- Arrays can have more than one dimension

```
string[,] s2D = new string[4,4];      // Makes a 4x4 array
s2D[0,0] = "A";
s2D[0,3] = "B";
s2D[1,2] = "C";
s2D[3,1] = "D";
```

  - This would make the 2-dimensional array

Multidimensional Arrays

- Arrays can have more than one dimension

```
string[,] s2D = new string[4,4];     // Makes a 4x4 array
s2D[0,0] = "A";
s2D[0,3] = "B";
s2D[1,2] = "C";
s2D[3,1] = "D";
```

  - This would make the 2-dimensional array

A			B
		C	
D			

Multidimensional Arrays

- Arrays can have more than one dimension

```
string[,] s2D = new string[4,4];     // Makes a 4x4 array
s2D[0,0] = "A";
s2D[0,3] = "B";
s2D[1,2] = "C";
s2D[3,1] = "D";
```

  - This would make the 2-dimensional array

```
| A |   |   | B |
|   |   | C |   |
|   |   |   |   |
| D |   |   |   |
```

  - Length is still the total length of the array

Multidimensional Arrays

- Arrays can have more than one dimension

```
string[,] s2D = new string[4,4];      // Makes a 4x4 array
s2D[0,0] = "A";
s2D[0,3] = "B";
s2D[1,2] = "C";
s2D[3,1] = "D";
```

  - This would make the 2-dimensional array

```
| A |   |   | B |
|   |   | C |   |
|   |   |   |   |
| D |   |   |   |
```

  - Length is still the total length of the array

```
print( s2D.Length );        // Prints: 16
```

Multidimensional Arrays

Multidimensional Arrays

```
string str = "";
for ( int i=0; i<4; i++ ) {
    for ( int j=0; j<4; j++ ) {
        if (s2D[i,j] != null) {
            str += "|" + s2D[i,j];
        } else {
            str += "|_";
        }
    }
    str += "|"+"\n";
}
print( str );
```

Multidimensional Arrays

```
string str = "";
for ( int i=0; i<4; i++ ) {
    for ( int j=0; j<4; j++ ) {
        if (s2D[i,j] != null) {
            str += "|" + s2D[i,j];
        } else {
            str += "|_";
        }
    }
    str += "|"+"\n";
}
print( str );
```

– This prints:

Multidimensional Arrays

```
string str = "";
for ( int i=0; i<4; i++ ) {
    for ( int j=0; j<4; j++ ) {
        if (s2D[i,j] != null) {
            str += "|" + s2D[i,j];
        } else {
            str += "|_";
        }
    }
    str += "|"+"\n";
}
print( str );
```

– This prints:

```
|A|_|_|B|
|_|_|C|_|
|_|_|_|_|
|D|_|_|_|
```

Jagged Lists & Arrays

Jagged Lists & Arrays

- Both Lists and arrays can be composed of other Lists or arrays

Jagged Lists & Arrays

- Both Lists and arrays can be composed of other Lists or arrays

```
string[][] jArray = new string[3][];     // Makes a 3x? array
```

Jagged Lists & Arrays

- **Both Lists and arrays can be composed of other Lists or arrays**

```
string[][] jArray = new string[3][];    // Makes a 3x? array
jArray[0] = new string[4];
jArray[0][0] = "A";
jArray[0][3] = "B";
```

Jagged Lists & Arrays

- Both Lists and arrays can be composed of other Lists or arrays

```
string[][] jArray = new string[3][];    // Makes a 3x? array
jArray[0] = new string[4];
jArray[0][0] = "A";
jArray[0][3] = "B";
jArray[1] = new string[] {"C","D","E"};
jArray[2] = new string[] {"F","G"};
```

Jagged Lists & Arrays

- **Both Lists and arrays can be composed of other Lists or arrays**

```
string[][] jArray = new string[3][];    // Makes a 3x? array
jArray[0] = new string[4];
jArray[0][0] = "A";
jArray[0][3] = "B";
jArray[1] = new string[] {"C","D","E"};
jArray[2] = new string[] {"F","G"};
```

  - This would make the jagged array

Jagged Lists & Arrays

- **Both Lists and arrays can be composed of other Lists or arrays**

```
string[][] jArray = new string[3][];    // Makes a 3x? array
jArray[0] = new string[4];
jArray[0][0] = "A";
jArray[0][3] = "B";
jArray[1] = new string[] {"C","D","E"};
jArray[2] = new string[] {"F","G"};
```

  - This would make the jagged array

```
| A |   |   | B |
| C | D | E |
| F | G |
```

Jagged Lists & Arrays

- **Both Lists and arrays can be composed of other Lists or arrays**

```
string[][] jArray = new string[3][];    // Makes a 3x? array
jArray[0] = new string[4];
jArray[0][0] = "A";
jArray[0][3] = "B";
jArray[1] = new string[] {"C","D","E"};
jArray[2] = new string[] {"F","G"};
```

  - This would make the jagged array

```
| A |   |   | B |
| C | D | E |
| F | G |
```

  - Length is now accurate for each part of the jagged array

Jagged Lists & Arrays

- **Both Lists and arrays can be composed of other Lists or arrays**

```
string[][] jArray = new string[3][];    // Makes a 3x? array
jArray[0] = new string[4];
jArray[0][0] = "A";
jArray[0][3] = "B";
jArray[1] = new string[] {"C","D","E"};
jArray[2] = new string[] {"F","G"};
```

  - This would make the jagged array

```
| A |   |   | B |
| C | D | E |
| F | G |
```

  - Length is now accurate for each part of the jagged array

```
print( jArray.Length );          // Prints: 4
print( jArray[1].Length );       // Prints: 3
```

`foreach` and Collections

`foreach` and Collections

- Lists and arrays can be iterated over using `foreach`

`foreach` and Collections

- Lists and arrays can be iterated over using `foreach`

```
string[] sArray = new string[] {"A","B","C","D"};
```

`foreach` and Collections

- Lists and arrays can be iterated over using `foreach`

```
string[] sArray = new string[] {"A","B","C","D"};
string str = "";
foreach (string s in sArray) {
    str += s;
}
print( s );                                    // Prints: "ABCD"
```

`foreach` and Collections

- Lists and arrays can be iterated over using `foreach`

```
string[] sArray = new string[] {"A","B","C","D"};
string str = "";
foreach (string s in sArray) {
    str += s;
}
print( s );                                    // Prints: "ABCD"

List<string> sList = new List<string>( sArray );
```

`foreach` and Collections

- Lists and arrays can be iterated over using `foreach`

```
string[] sArray = new string[] {"A","B","C","D"};
string str = "";
foreach (string s in sArray) {
    str += s;
}
print( s );                                          // Prints: "ABCD"

List<string> sList = new List<string>( sArray );
string str2 = "";
foreach (string s in sList) {
    str2 += s;
}
print( s2 );                                         // Prints: "ABCD"
```

`foreach` and Collections

- Lists and arrays can be iterated over using `foreach`

```
string[] sArray = new string[] {"A","B","C","D"};
string str = "";
foreach (string s in sArray) {
    str += s;
}
print( s );                                    // Prints: "ABCD"

List<string> sList = new List<string>( sArray );
string str2 = "";
foreach (string s in sList) {
    str2 += s;
}
print( s2 );                                   // Prints: "ABCD"
```

- Why can `string s` be declared twice?

`foreach` and Collections

- Lists and arrays can be iterated over using `foreach`

```
string[] sArray = new string[] {"A","B","C","D"};
string str = "";
foreach (string s in sArray) {
    str += s;
}
print( s );                                    // Prints: "ABCD"

List<string> sList = new List<string>( sArray );
string str2 = "";
foreach (string s in sList) {
    str2 += s;
}
print( s2 );                                   // Prints: "ABCD"
```

- Why can `string s` be declared twice?
  - Because `string s` is local to each `foreach` loop

When to Use List or Array

When to Use List or Array

- Each have pros and cons:

When to Use List or Array

- **Each have pros and cons:**
  - **List has flexible length, whereas array length is more difficult to change.**

When to Use List or Array

- **Each have pros and cons:**
  - **List has flexible length, whereas array length is more difficult to change.**
  - **Array is very slightly faster.**

When to Use List or Array

- **Each have pros and cons:**
  - List has flexible length, whereas array length is more difficult to change.
  - Array is very slightly faster.
  - Array allows multidimensional indices.

When to Use List or Array

- **Each have pros and cons:**
  - List has flexible length, whereas array length is more difficult to change.
  - Array is very slightly faster.
  - Array allows multidimensional indices.
  - Array allows empty elements in the middle of the collection.

When to Use List or Array

- **Each have pros and cons:**
  - List has flexible length, whereas array length is more difficult to change.
  - Array is very slightly faster.
  - Array allows multidimensional indices.
  - Array allows empty elements in the middle of the collection.
- **Because they are simpler to implement and take less forethought (due to their flexible length), the author tends to use Lists much more often than arrays.**

When to Use List or Array

- **Each have pros and cons:**
  - List has flexible length, whereas array length is more difficult to change.
  - Array is very slightly faster.
  - Array allows multidimensional indices.
  - Array allows empty elements in the middle of the collection.
- **Because they are simpler to implement and take less forethought (due to their flexible length), the author tends to use Lists much more often than arrays.**
  - This is especially true when prototyping games, since prototyping requires a lot of flexibility.

Other Collection Types

Other Collection Types

- ArrayList

Other Collection Types

- **ArrayList**
  - **ArrayList is like a List except without a set type**

Other Collection Types

- **ArrayList**
  - ArrayList is like a List except without a set type
  - Extremely flexible, but I find them less useful and slower

Other Collection Types

- ArrayList
  - ArrayList is like a List except without a set type
  - Extremely flexible, but I find them less useful and slower
- Dictionary<key,value>

Other Collection Types

- ArrayList
  - ArrayList is like a List except without a set type
  - Extremely flexible, but I find them less useful and slower
- Dictionary<key,value>
  - A key / value pair

Other Collection Types

- **ArrayList**
  - **ArrayList is like a List except without a set type**
  - **Extremely flexible, but I find them less useful and slower**
- **Dictionary<key,value>**
  - **A key / value pair**
    - The key could be a string, like a username

Other Collection Types

- **ArrayList**
  - **ArrayList is like a List except without a set type**
  - **Extremely flexible, but I find them less useful and slower**
- **Dictionary<key,value>**
  - **A key / value pair**
    - The key could be a string, like a username
    - The value would be the user data

Other Collection Types

- **ArrayList**
  - **ArrayList is like a List except without a set type**
  - **Extremely flexible, but I find them less useful and slower**
- **Dictionary<key,value>**
  - **A key / value pair**
    - The key could be a string, like a username
    - The value would be the user data
  - **Used in some of the later tutorials in the book**

Other Collection Types

- **ArrayList**
  - **ArrayList is like a List except without a set type**
  - **Extremely flexible, but I find them less useful and slower**
- **Dictionary<key,value>**
  - **A key / value pair**
    - The key could be a string, like a username
    - The value would be the user data
  - **Used in some of the later tutorials in the book**
  - **Requires `using System.Collections.Generic;`**

Other Collection Types

- **ArrayList**
  - **ArrayList is like a List except without a set type**
  - **Extremely flexible, but I find them less useful and slower**
- **Dictionary<key,value>**
  - **A key / value pair**
    - The key could be a string, like a username
    - The value would be the user data
  - **Used in some of the later tutorials in the book**
  - **Requires `using System.Collections.Generic;`**
  - **Very useful**

Other Collection Types

- **ArrayList**
  - **ArrayList is like a List except without a set type**
  - **Extremely flexible, but I find them less useful and slower**
- **Dictionary<key,value>**
  - **A key / value pair**
    - The key could be a string, like a username
    - The value would be the user data
  - **Used in some of the later tutorials in the book**
  - **Requires `using System.Collections.Generic;`**
  - **Very useful**
  - **But don't appear properly in the Unity Inspector**

Chapter 22 – Summary

Chapter 22 – Summary

- The collections we use in this book can hold any number of objects (of a single type)

Chapter 22 – Summary

- The collections we use in this book can hold any number of objects (of a single type)
- List is the most easily usable collection type

Chapter 22 – Summary

- The collections we use in this book can hold any number of objects (of a single type)
- List is the most easily usable collection type
  - It requires `using System.Collections.Generic;`

Chapter 22 – Summary

- The collections we use in this book can hold any number of objects (of a single type)
- List is the most easily usable collection type
  - It requires `using System.Collections.Generic;`
- Arrays are less flexible but also very useful

Chapter 22 – Summary

- The collections we use in this book can hold any number of objects (of a single type)
- List is the most easily usable collection type
  - It requires `using System.Collections.Generic;`
- Arrays are less flexible but also very useful
  - Only arrays can be multidimensional

Chapter 22 – Summary

- The collections we use in this book can hold any number of objects (of a single type)
- List is the most easily usable collection type
  - It requires `using System.Collections.Generic;`
- Arrays are less flexible but also very useful
  - Only arrays can be multidimensional
- Both Lists and arrays can be jagged

Chapter 22 – Summary

- The collections we use in this book can hold any number of objects (of a single type)
- List is the most easily usable collection type
  - It requires `using System.Collections.Generic;`
- Arrays are less flexible but also very useful
  - Only arrays can be multidimensional
- Both Lists and arrays can be jagged
- I generally recommend Lists over arrays

Chapter 22 – Summary

- The collections we use in this book can hold any number of objects (of a single type)
- List is the most easily usable collection type
  - It requires `using System.Collections.Generic;`
- Arrays are less flexible but also very useful
  - Only arrays can be multidimensional
- Both Lists and arrays can be jagged
- I generally recommend Lists over arrays
- Dictionaries will also be used in this book

Chapter 22 – Summary

- The collections we use in this book can hold any number of objects (of a single type)
- List is the most easily usable collection type
  - It requires `using System.Collections.Generic;`
- Arrays are less flexible but also very useful
  - Only arrays can be multidimensional
- Both Lists and arrays can be jagged
- I generally recommend Lists over arrays
- Dictionaries will also be used in this book
- Next Chapter: Functions!