Beginning C# Collections

INTRODUCING COLLECTIONS AND ARRAYS

Simon Robinson
LEAD SOFTWARE DEVELOPER

@TechieSimon   www.SimonRobinson.com

Overview

Collections for multiple items

Coding with arrays

- Enumerate data
- Look-up/replace data
- Zero-based indexing

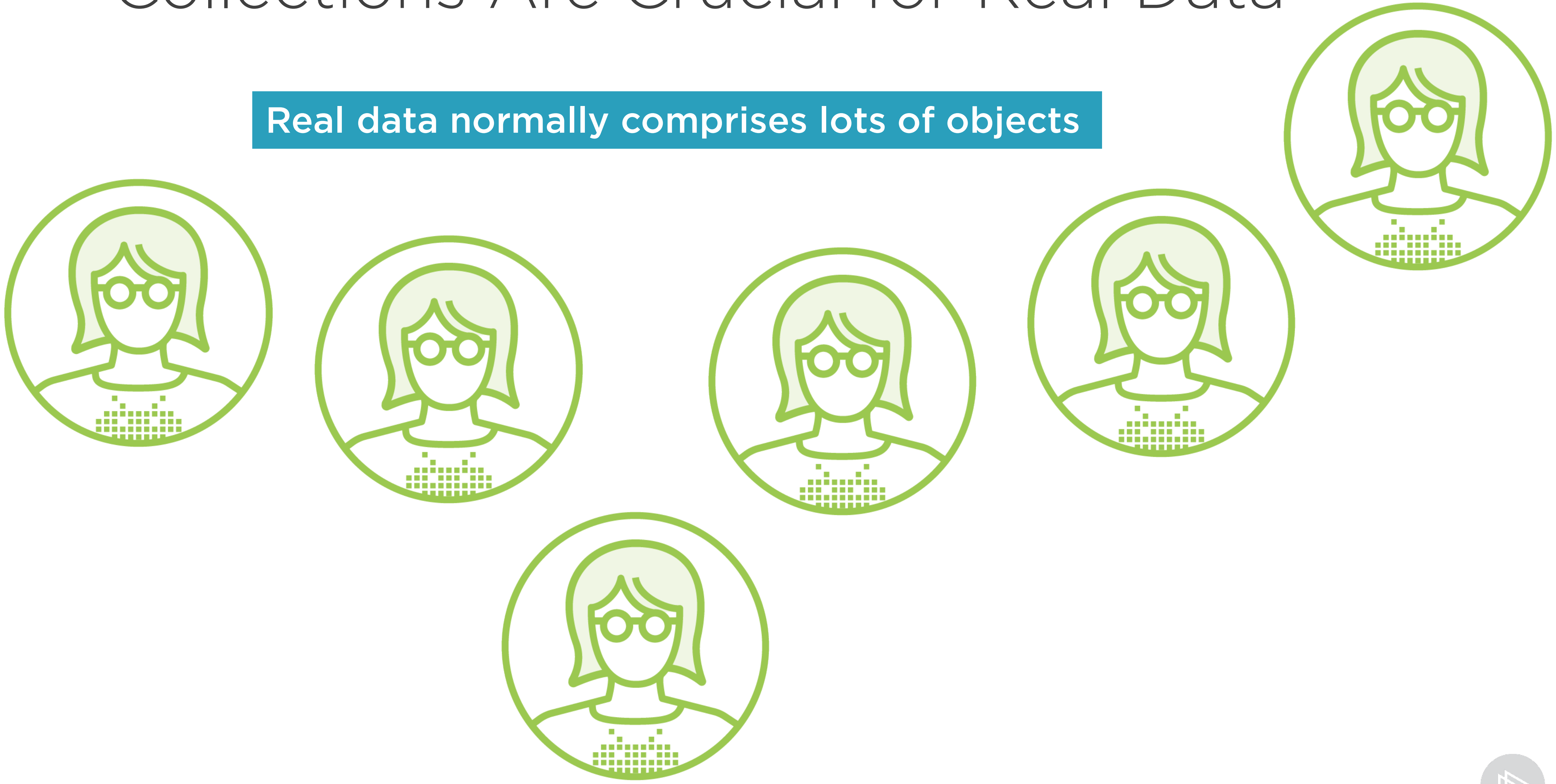
Collection debugging

Collection

A type whose purpose is to group data together.

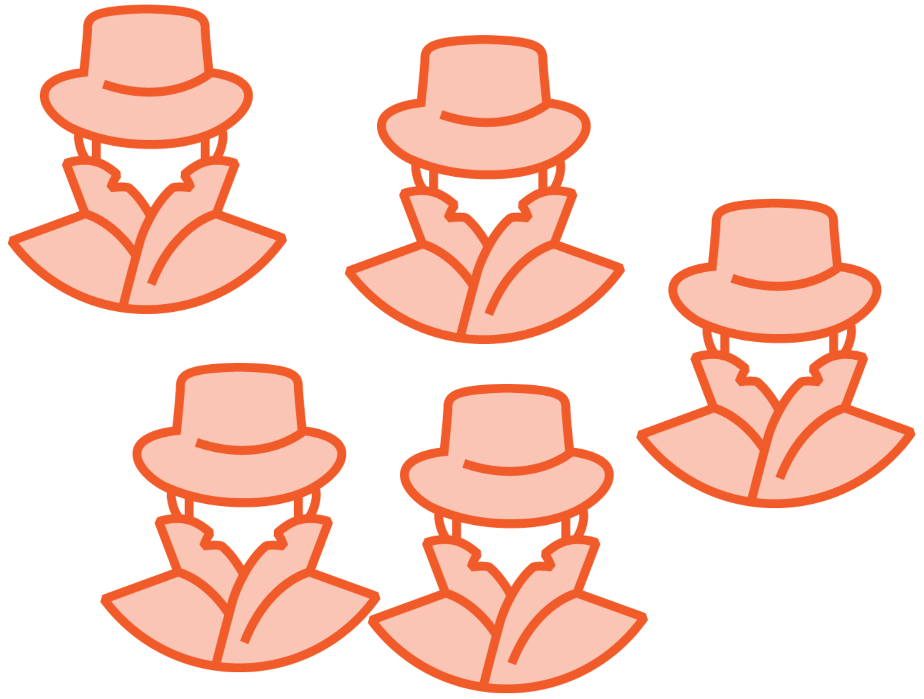
Collections Are Crucial for Real Data

Real data normally comprises lots of objects



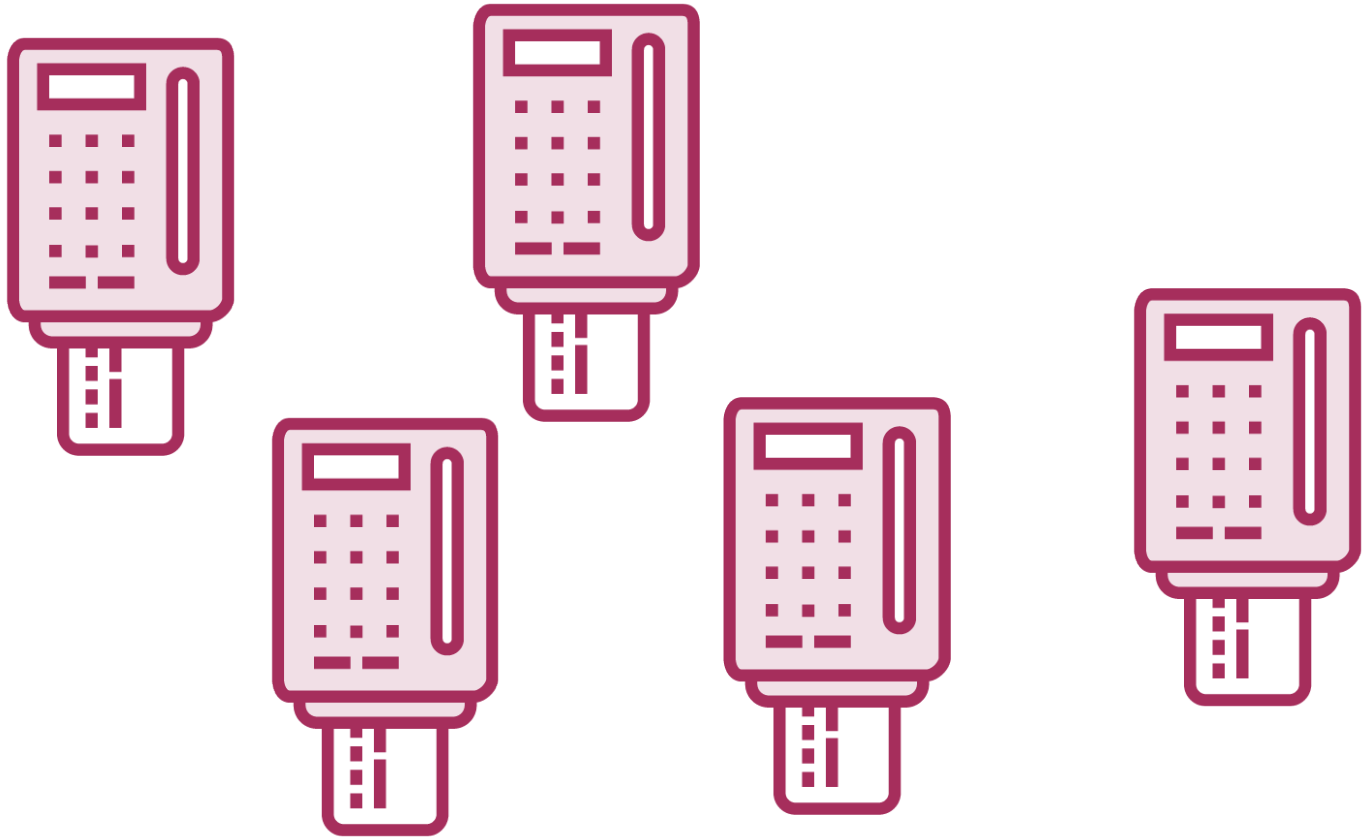
Collections Are Crucial for Real Data

Real data normally comprises lots of objects



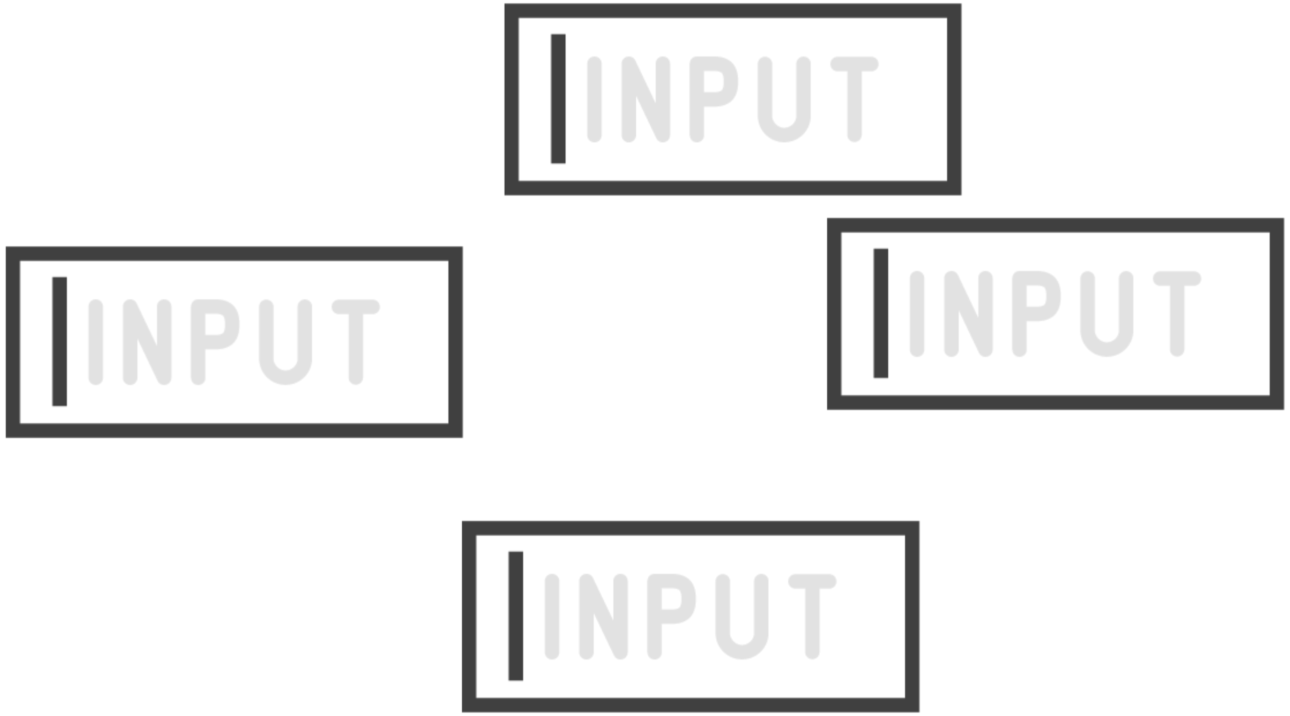
Collections Are Crucial for Real Data

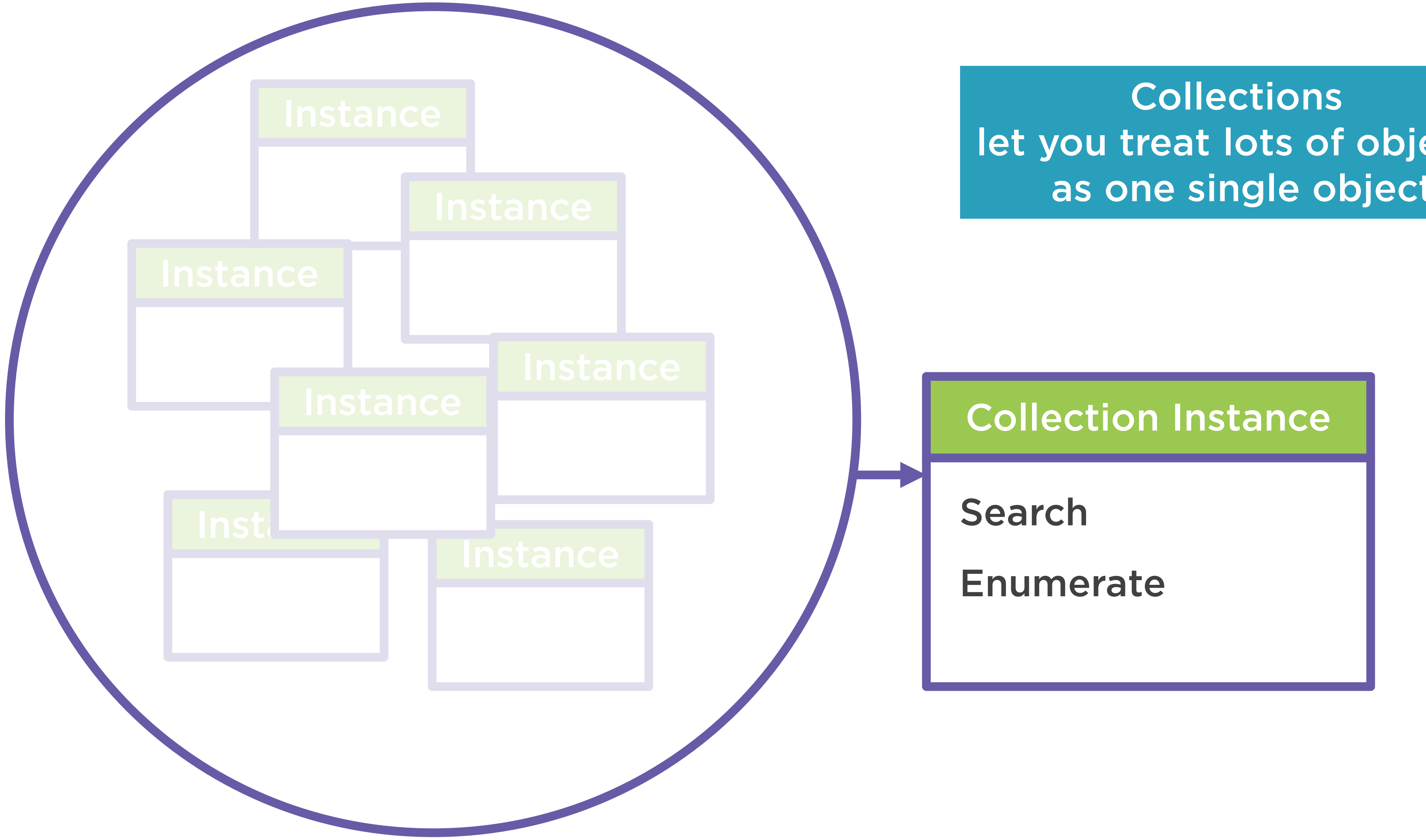
Real data normally comprises lots of objects



Collections Are Crucial for Real Data

Real data normally comprises lots of objects





Collections
let you treat lots of objects
as one single object

Three Collections…

Array

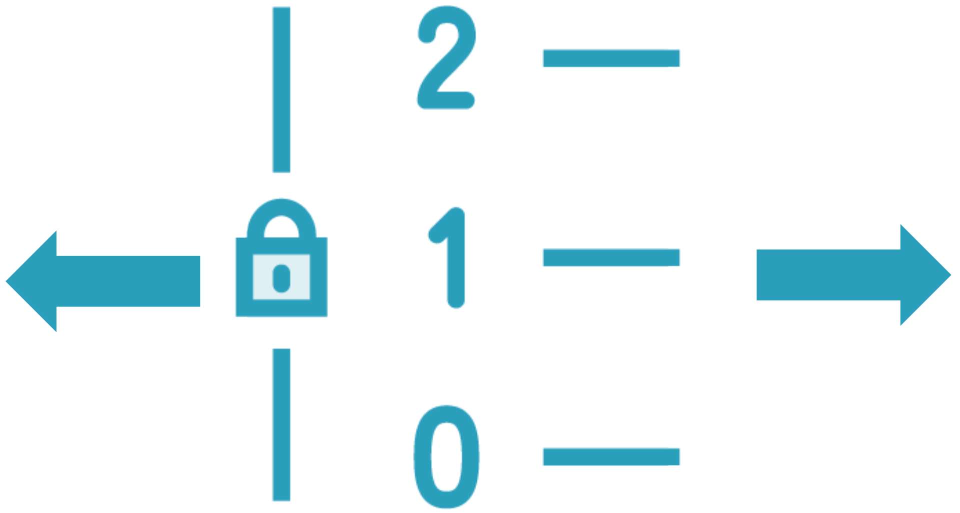
List

Dictionary

Most widely used general purpose collections

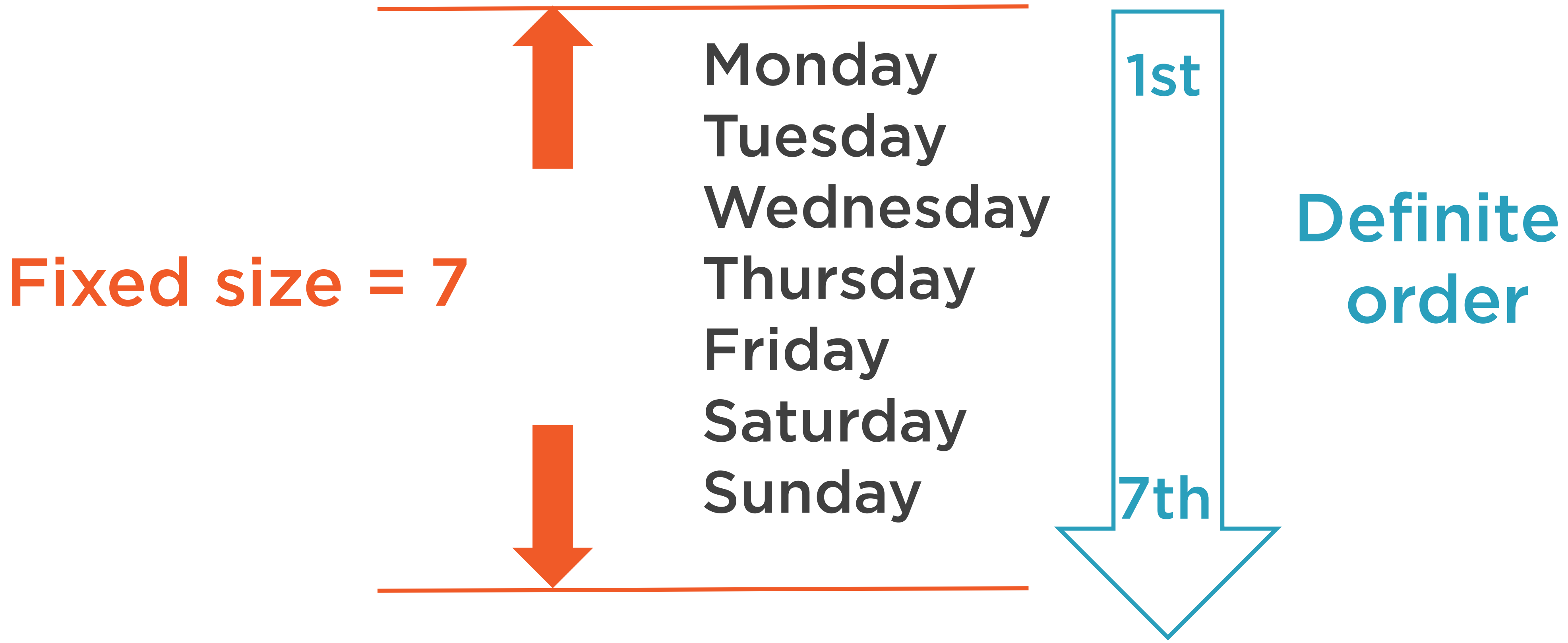
Array Characteristics

Fixed size



Ordered

Days of the Week



Demo

Example: Days of the week

Instantiate an array

Display all items

Code Demo

You Can Make an Array of Anything

Array of int

```
int[] ints = { 1, 4, 9 };
```

Array of Point

```
System.Drawing.Point[] points =
{
    new System.Drawing.Point(3, 5),
    // etc.
};
```

Some Terminology

Element or **Item**

An object (or struct) in a collection

Enumerate or **Iterate**

Go through each item in turn

Look up an item

Access an individual item in a collection

Get day by number....

1 →	Monday
2 →	Tuesday

Demo

n’th day demo

- Looking up items

Code Demo

Arrays Are Zero-indexed

Human (1-based) indexing

Zero-based indexing

1	Monday	0
2	Tuesday	1
3	Wednesday	2
4	Thursday	3
5	Friday	4
6	Saturday	5
7	Sunday	6

2
1
0

Code Demo

Code Demo

Monday	0
Tuesday	1
Wednesday	2
Thursday	3
Friday	4
Saturday	5
Sunday	6

Zero-based indexing

Code Demo

Why Use Zero-based Indexing?

Historical reasons

Made memory management easier

Better for performance (when computers were slow)

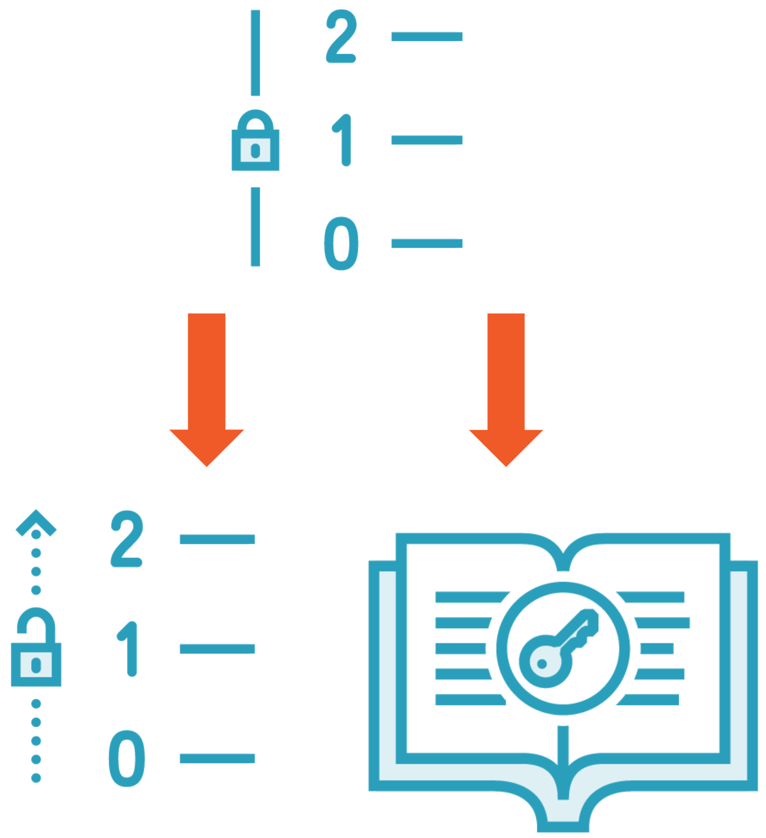
Demo

Modify an array

- Replace items

Code Demo

Arrays to Other Collections



Same principles for most collections

- Square bracket look-up syntax
- foreach loop
- Zero-based indexing
- Debugger integration

Summary

Collections group items together

`foreach` to enumerate

`[]` to identify items

Zero-based indexing

Collections integrate into debugger

Importing from a Data Source into an Array

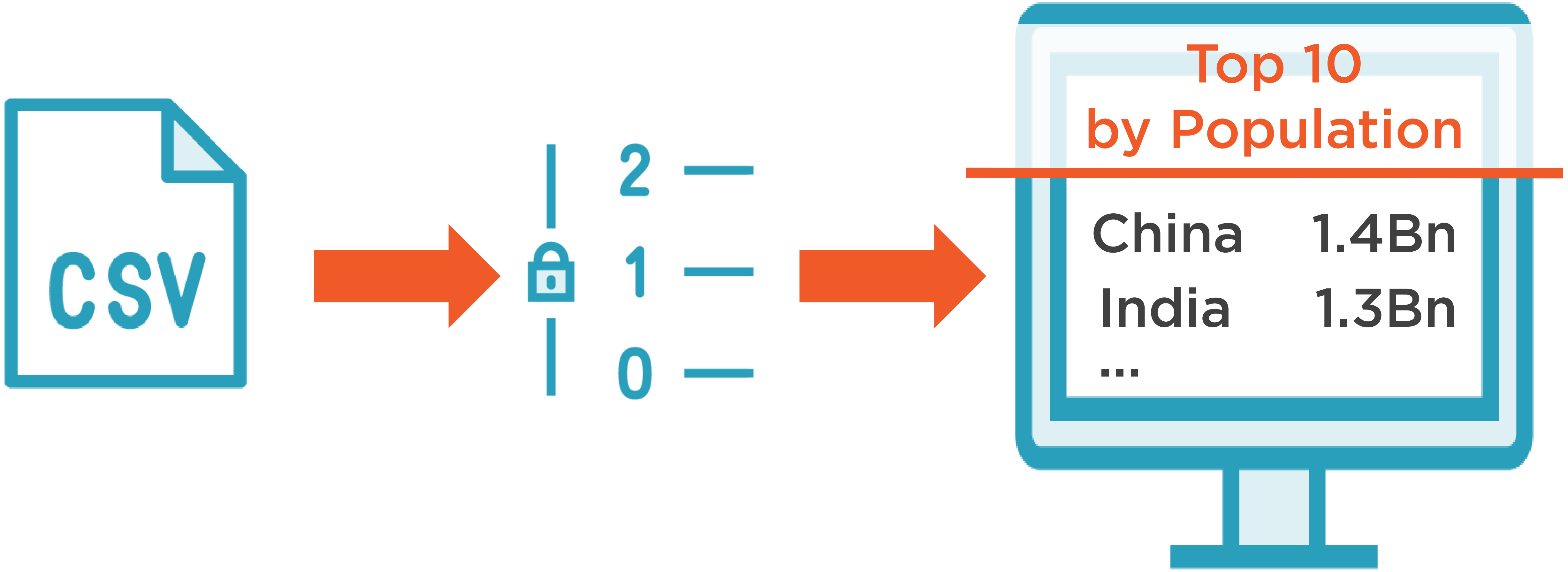
Simon Robinson
LEAD SOFTWARE DEVELOPER
@TechieSimon www.SimonRobinson.com

Overview

Introduce sample demo

- Dynamically put data in an array
- Uninitialized array contains `nulls`
- Arrays are ubiquitous

Countries Demo App



Demo

Read top 10 countries from CSV file

CSV/web Demo

Previously...

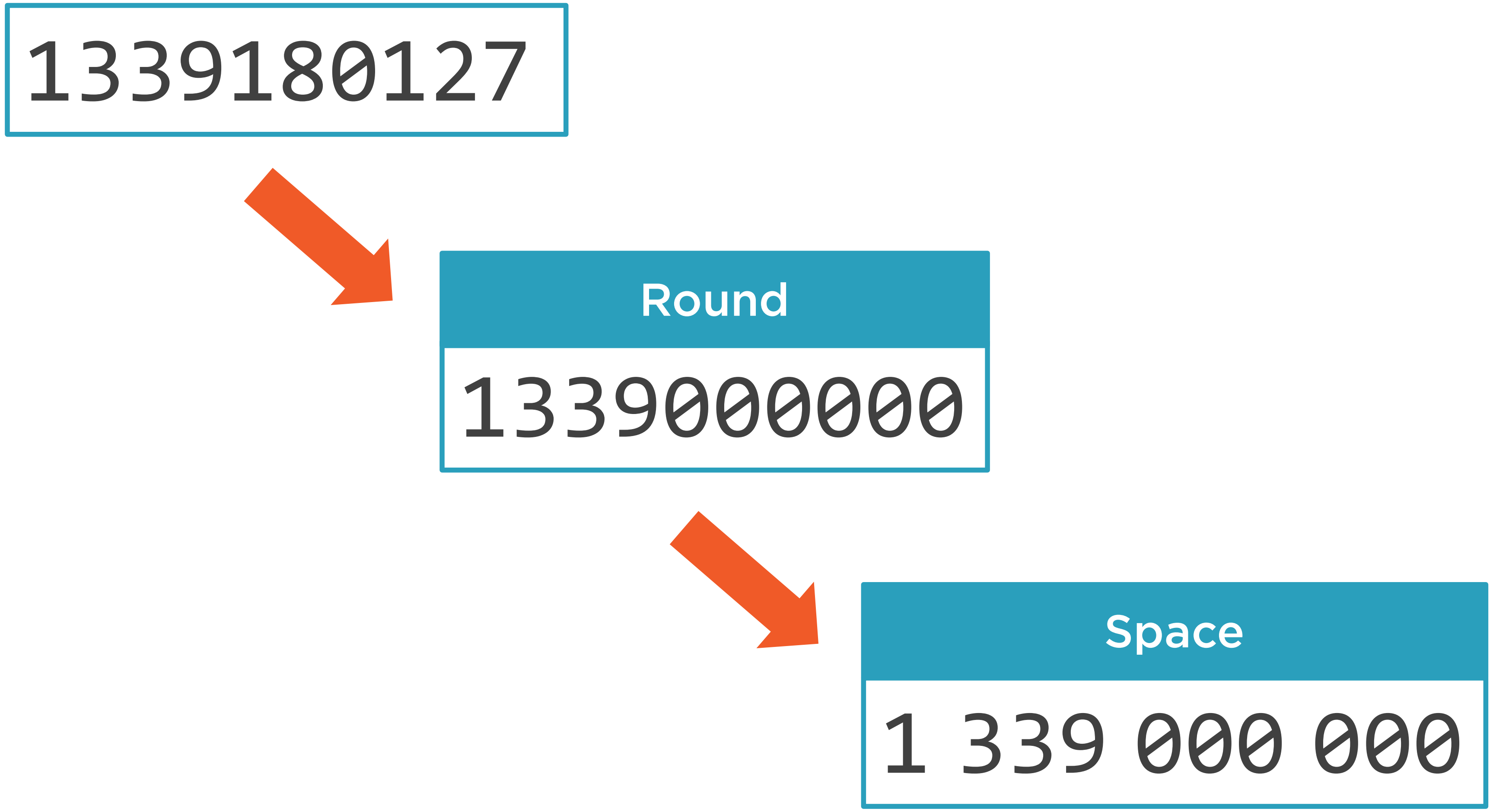
```
string[] daysOfWeek = {
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday",
        "Sunday"
};
```

This is a
Collection Initializer
(Array initializer)

Can’t do this
if you don’t know
the values
when the array is instantiated

CSV/web Demo

Formatting Population



Demos

Instantiating an Array

```
Country[] countries = null;
```

Declaring, not instantiating

Arrays Are **Always** Reference Types

✓

```
// OK. int[] is a reference type
int[] numbers = null;
```

✗

```
// Wrong. int is a value type so can't be null
int number = null;
```

Instantiating an Array

```
// nValues is an int
Country[] countries = new Country[nValues];
```

`countries` will contain all nulls

Minimum information you must provide

```
// nValues is an int
int[] ints = new int[nValues];
```

`ints` will contain all zeros

Instantiating an Array

```
// country1, country2 etc. are of type Country
Country[] countries = ~~new Country[]~~
{
        country1, country2, country3, country4
}
```

Specifying all values

Summary

Demo: Import data into an array

- Initialize an array by size
- Array starts full of `null`/default values
- Can populate with `for` loop
- Arrays used in .NET Framework

Selecting Items Using LINQ

Simon Robinson
LEAD SOFTWARE DEVELOPER

@TechieSimon   www.SimonRobinson.com

Overview

Querying: Extracting the data you want

LINQ is read-only

LINQ query syntax

LINQ vs. loops

Demo

Limit how many elements you enumerate

- With a for loop in last module
- Now using LINQ

Code Demo

Demo

Re-order list elements

- Display countries in alphabetical order

Code Demo

What Is LINQ Doing?

```
foreach (Country country in countries.Take(10).OrderBy(x=>x.Name))
```

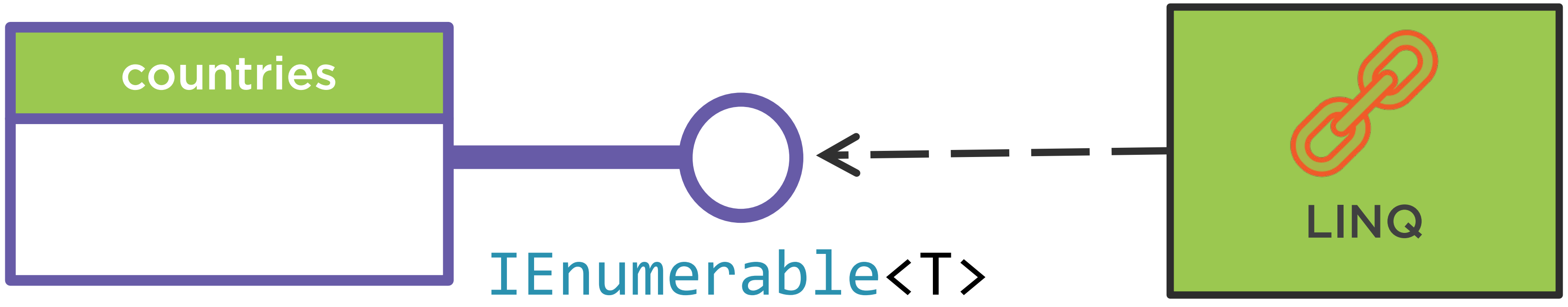
To understand LINQ,
don’t think of countries as a collection!

What Is LINQ Doing?

```
foreach (Country country in countries.Take(10).OrderBy(x=>x.Name))
```

To LINQ, countries is just a data source

Something that can be enumerated



LINQ Passes on Objects

```
foreach (Country country in countries.Take(10).OrderBy(x=>x.Name))
```

Take()

Grabs values…

… counts…

… and passes them on

OrderBy()

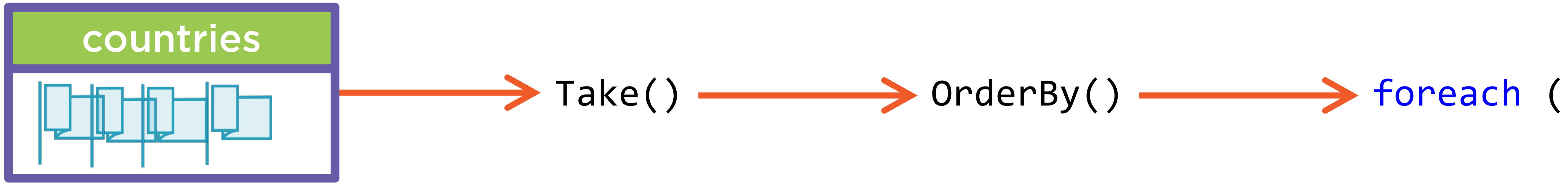
Grabs values…

… sorts…

… and passes them on

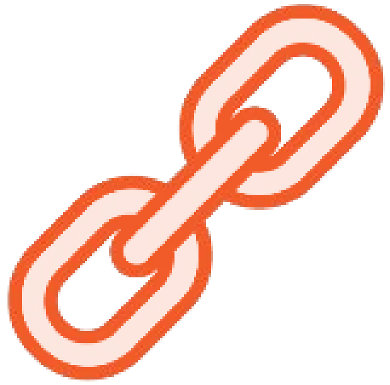
LINQ Passes on Objects

```
foreach (Country country in countries.Take(10).OrderBy(x=>x.Name))
```



This chain queries the data in the collection

Think of LINQ as...



Steps in a chain

Queries

Code Demo

LINQ Is for All Collections

for loop	LINQ
Arrays and lists only	All collections
(Because requires an index)	(including dictionaries)

Removing Items from a Collection

Example: Removing countries with commas

```
// for loop
if (country.Name.Contains(','))
{
    countries.RemoveAt(i);
    // etc
```

```
// RemoveAll()
countries.RemoveAll
     (x => x.Name.Contains(','));
```

LINQ Can't do this
(LINQ is read-only)

Demo

Receive all items from collection

- But filter them
- View list of countries without commas

Code Demo

Demo

LINQ Query Syntax

Code Demo

LINQ Query Syntax

✓ Complex queries can be more readable

✗ New syntax to learn

Doesn’t support all LINQ features

LINQ

Language Integrated Query

Three Techniques

LINQ

for loop

Collection methods

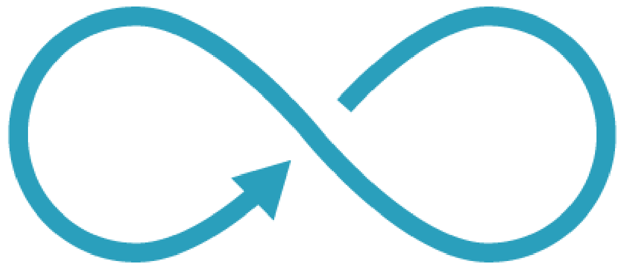
LINQ

Very simple code

Only for querying – not modifying

Good for productivity

for Loop



Very flexible

Only for ordered collections

Most complex to code

Collection Methods

Specific to a few tasks

Mainly for Array and List

Simple and efficient

Summary

LINQ

- Treats collections as enumerables
- `Take`, `OrderBy`, and `Where`
- Queries collections, doesn’t modify
- LINQ query syntax

Creating Collections of Collections

Simon Robinson
LEAD SOFTWARE DEVELOPER

@TechieSimon   www.SimonRobinson.com

Overview

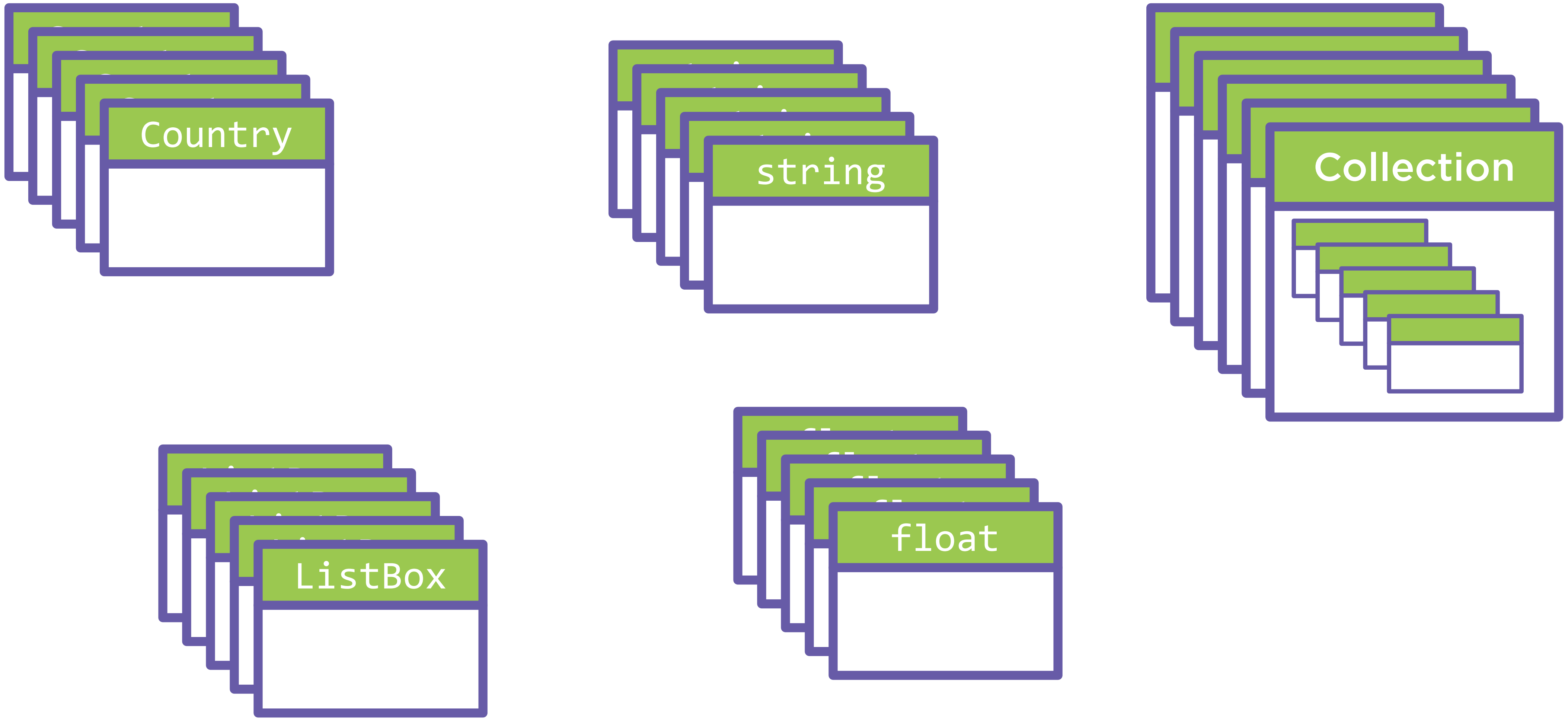
Collections of collections

- Putting collections in other collections

And combining what you've learned…

- Arrays
- Lists
- Dictionaries
  - Keys
- for loops
- LINQ

Collections Can Contain Anything



Display Countries in a Region

User types in:	Africa
App displays:	Nigeria Ethiopia Egypt, Arab Rep. etc.

Display Countries in a Region

User types in: South America

App displays: Brazil
Colombia
Argentina
etc.

What kind of collection?

What Kind of Collection?

South America

Brazil

Colombia

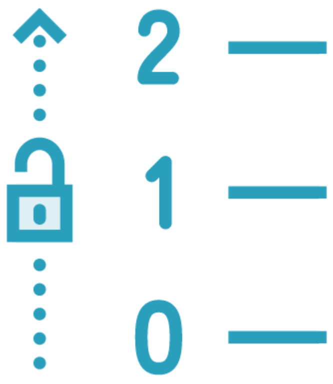
Argentina

etc.

```
Dictionary<string,                    >
```

What Kind of Collection?

Array



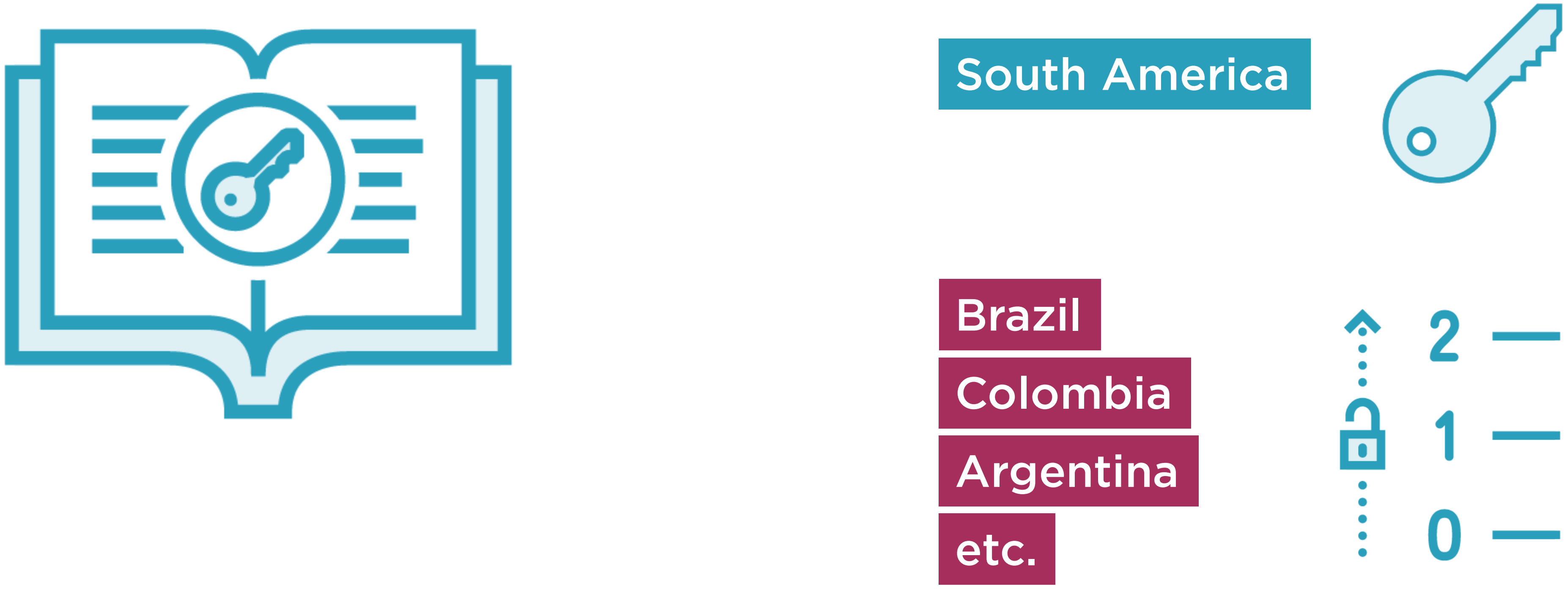
List

Dictionary

Requirements: Ordered

Dynamically sized

What Kind of Collection?



```
Dictionary<string, List<Country>>
```

What Kind of Collection?

This partitions the list of countries

```
Dictionary<string, List<Country>>
```

Demo

Display top 10 countries in any region

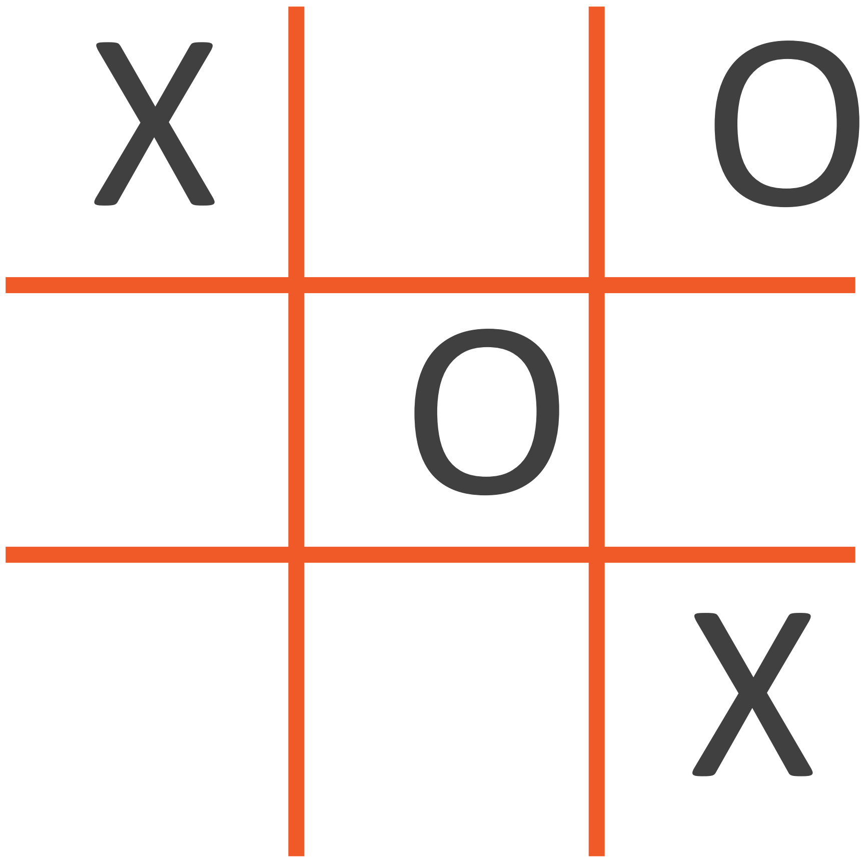
- Demos nested collection
- Demos manipulating keys

Code Demo

Arrays

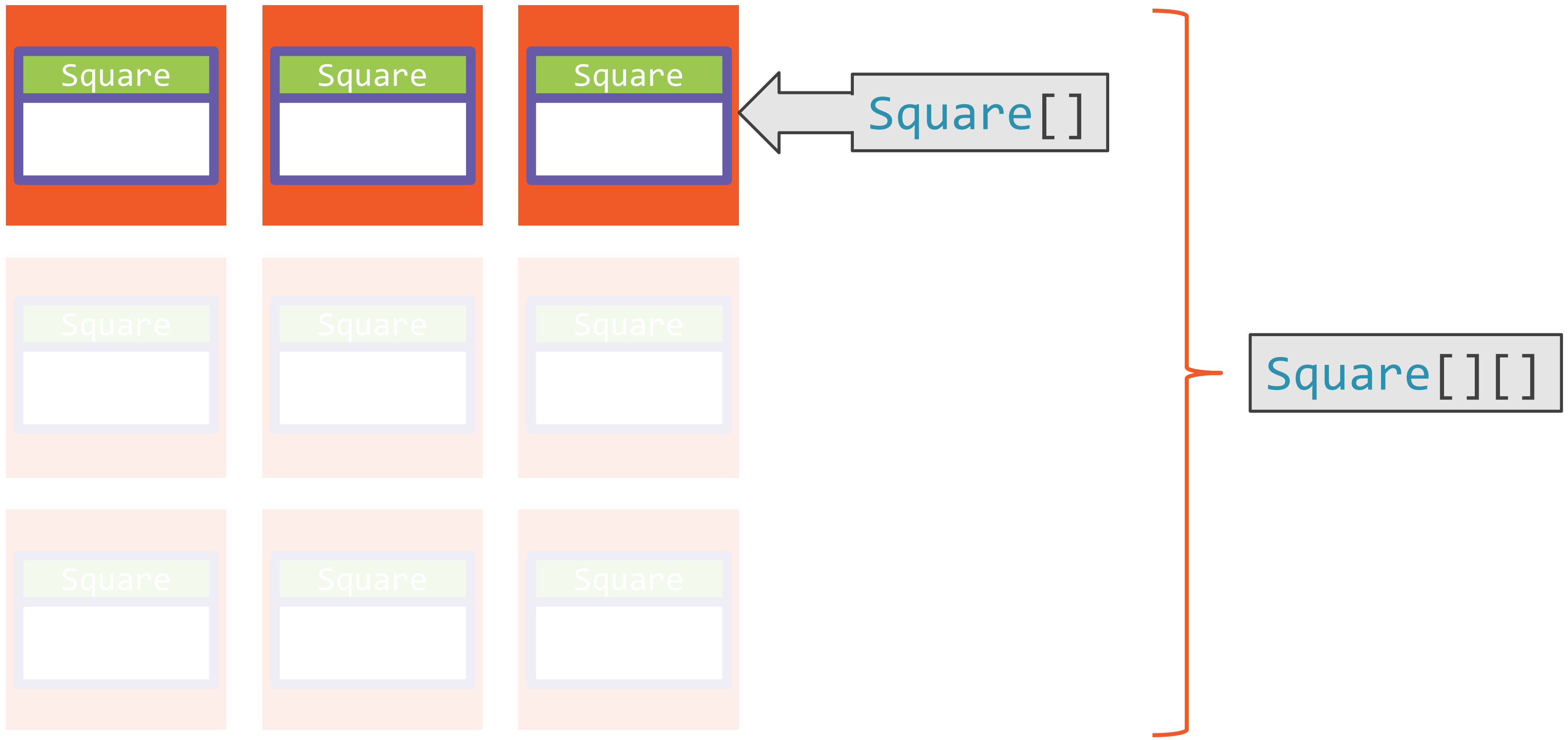
Arrays of arrays gives particularly simple syntax

Noughts and Crosses



(Tic-tac-toe in some countries)

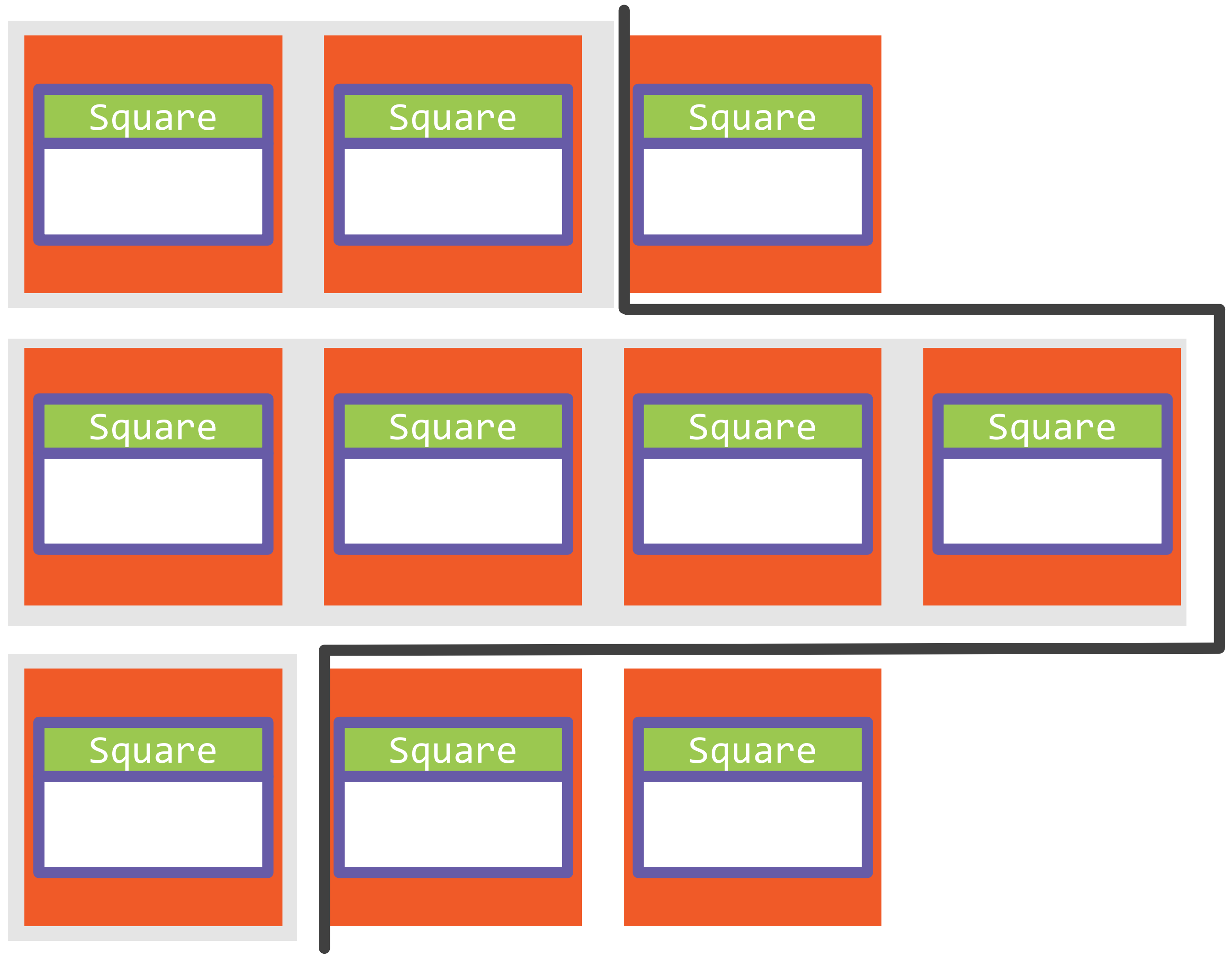
Noughts and Crosses



Jagged array

Array of arrays.

Jagged Array



Demo

Noughts and Crosses Game

Code Demo

Some Terminology

Jagged Array

T[][]

Array of arrays

Multidimensional array

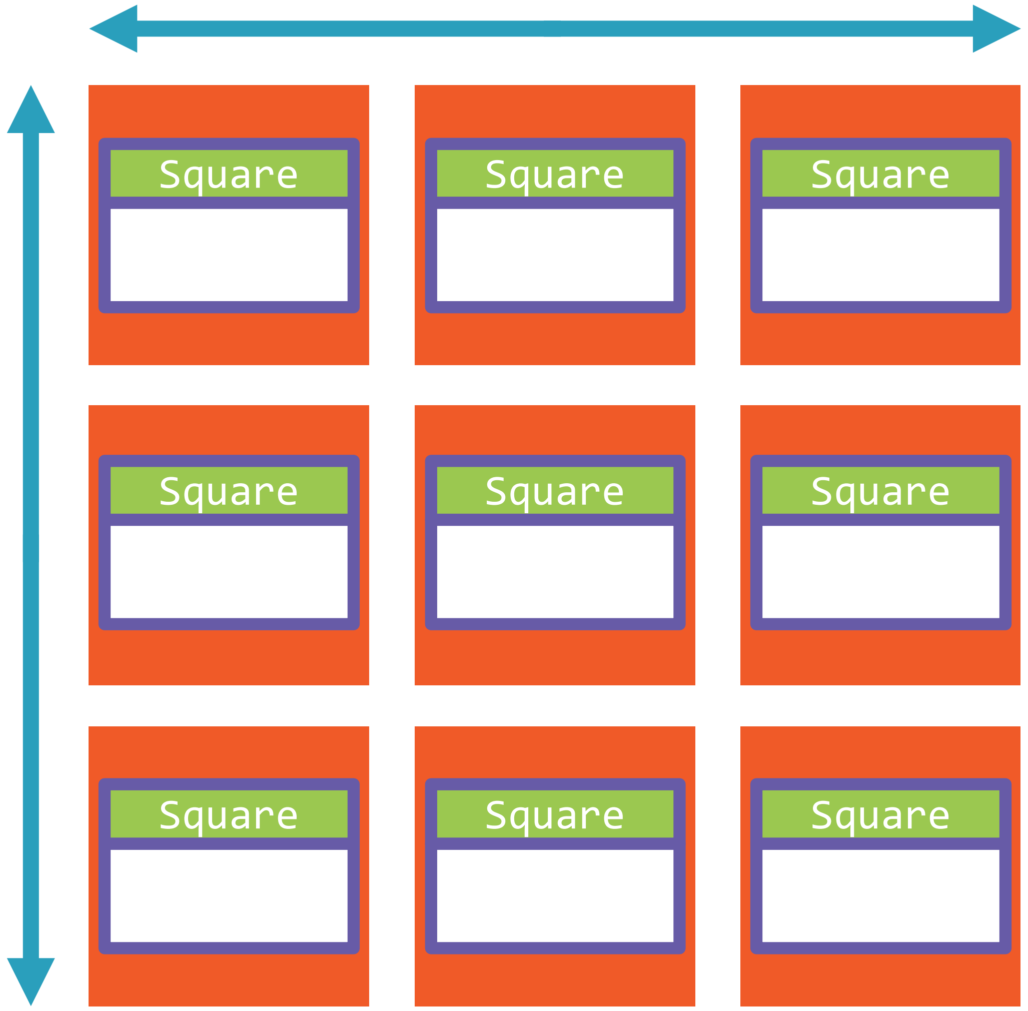
T[,]

Single array with two indices

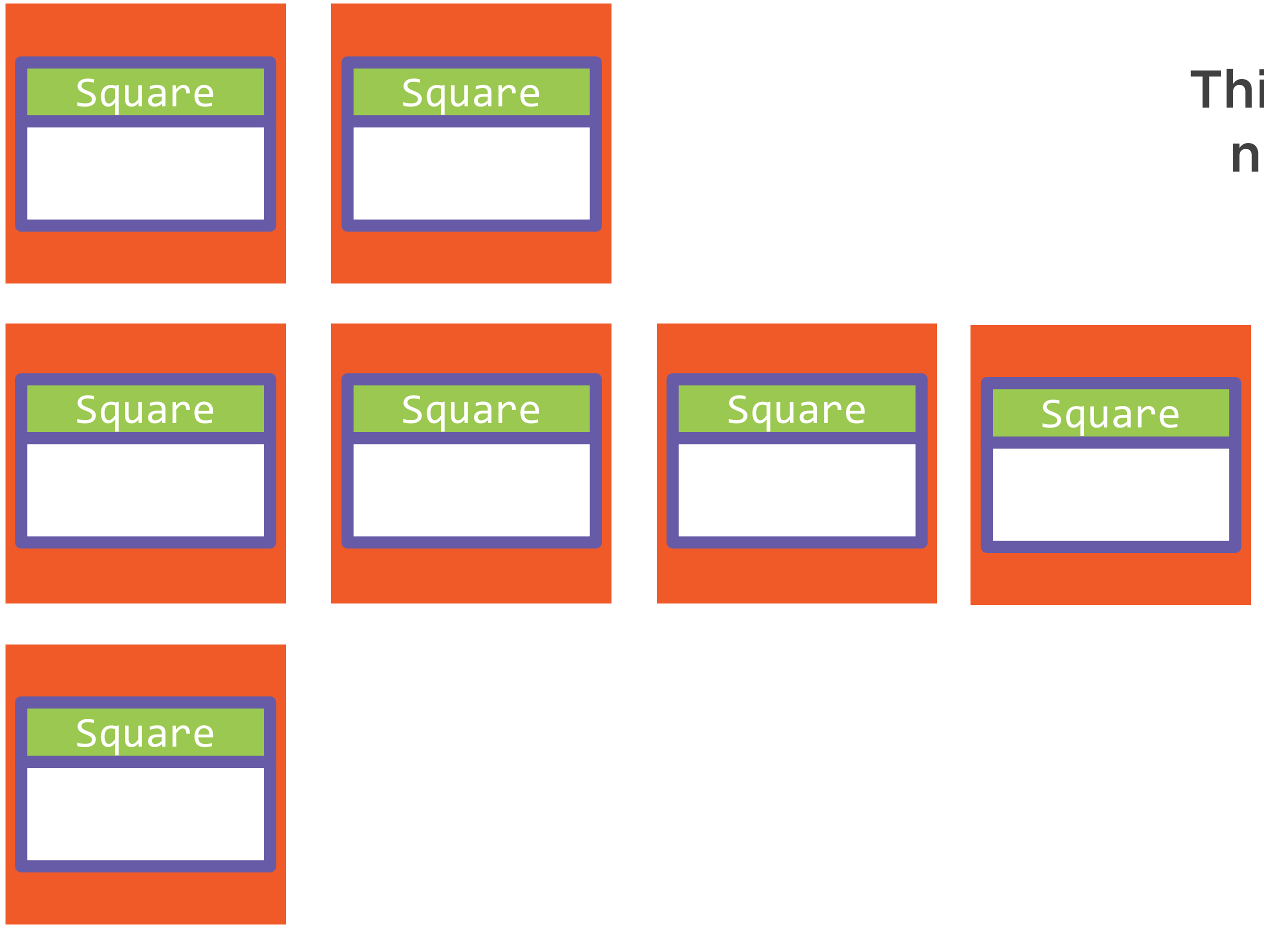
Code Demo

Multidimensional Arrays

Require a completely regular grid – cannot be jagged



Jagged Arrays



This can only be a jagged array, not a multidimensional array

Summary

Collections of collections

- `Allows partitioning data`
- `Chained look-ups: T[][]`
- `Jagged arrays`
- `Multidimensional arrays`

Resizing Collections with Lists

Simon Robinson
LEAD SOFTWARE DEVELOPER

@TechieSimon www.SimonRobinson.com

Overview

List<T>

- Can be resized
- Useful when how many items unknown at instantiation
- Similar to arrays in coding
- Searching

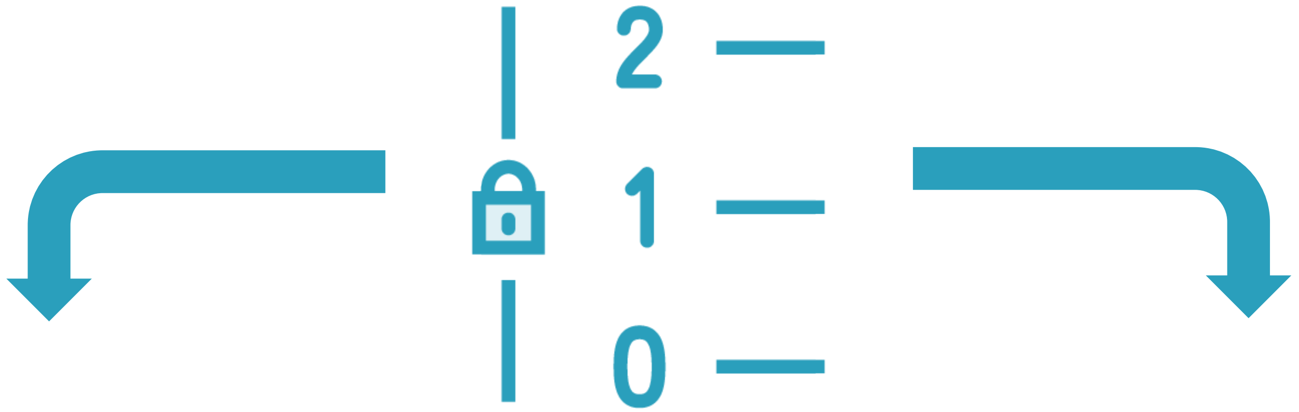
Code Demo

Arrays

How do you instantiate without knowing the number of elements?

- You can't!
- Can never change the size after instantiation

Arrays



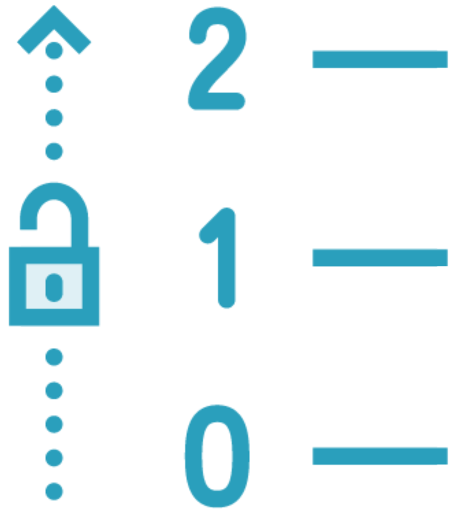
✓

Great for fixed size data

✗

Not good if you don’t know the size before reading the data

List<T>



Similar to arrays

Except resizable

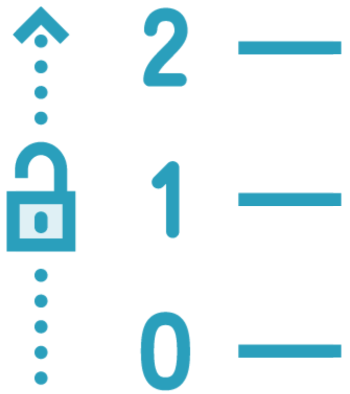
Demo

Basics of List<T> coding

- Days of the week

Code Demo

Lists vs. Arrays



List<T>	T[]
More flexible	Simpler syntax

Code Demo

List<T> and Generics

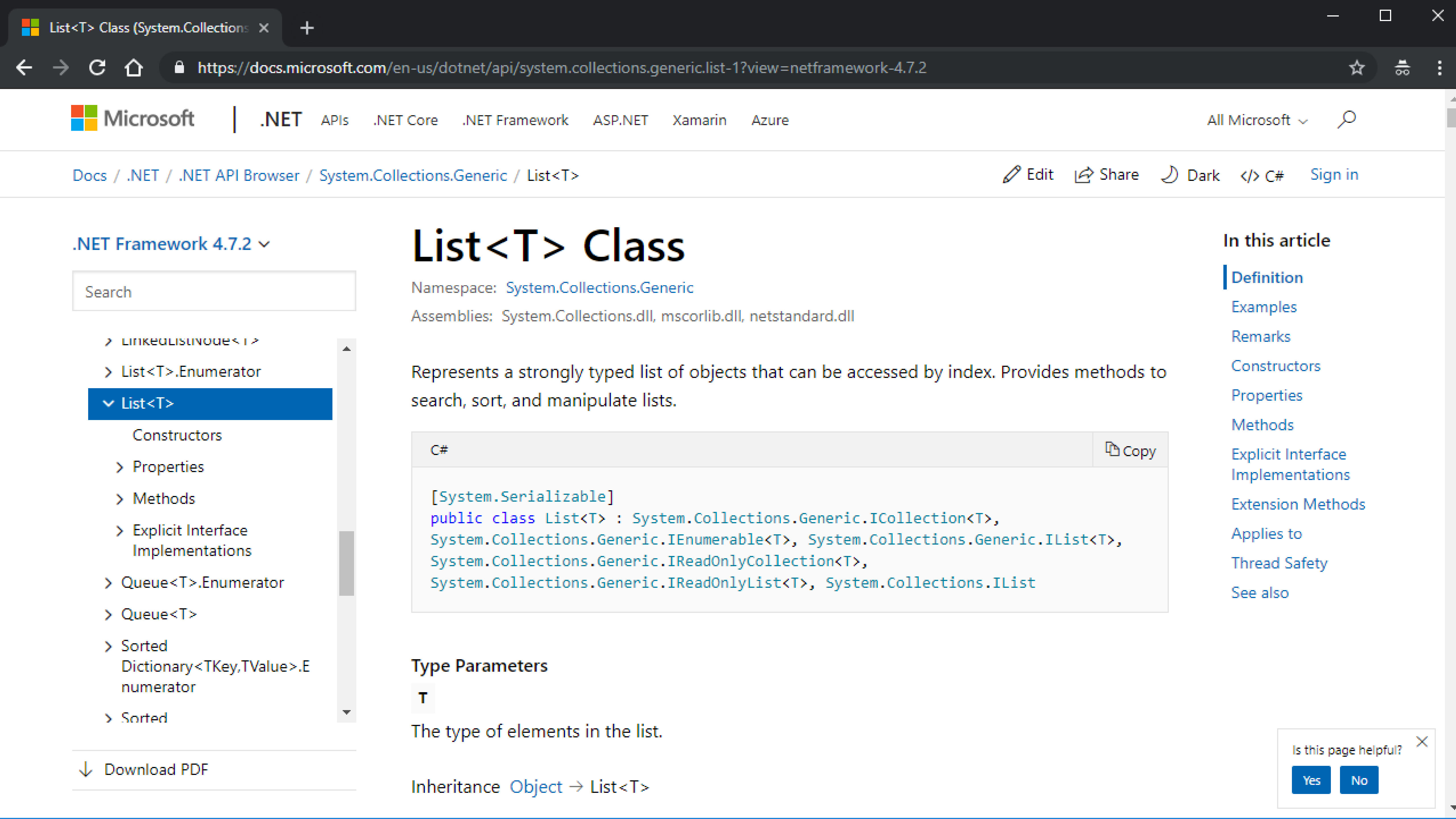
`List<T>`

Angle brackets indicate a generic type

Simplified version:
The type you're storing in the collection goes in angle brackets

(Except arrays: `T[]`)

Use T to refer to an unspecified type



List<T> Class

Namespace: System.Collections.Generic

Assemblies: System.Collections.dll, mscorlib.dll, netstandard.dll

Represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists.

C#

```
[System.Serializable]
public class List<T> : System.Collections.Generic.ICollection<T>,
System.Collections.Generic.IEnumerable<T>, System.Collections.Generic.IList<T>,
System.Collections.Generic.IReadOnlyCollection<T>,
System.Collections.Generic.IReadOnlyList<T>, System.Collections.IList
```

Type Parameters

T

The type of elements in the list.

Inheritance Object → List<T>

Demo

Last module: Imported 10 countries from CSV

Now: Import ALL countries from CSV

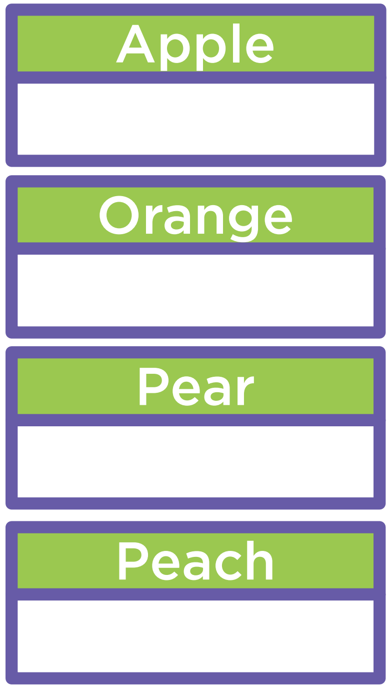
Code Demo

Adding and Inserting

Adding

Append to end of list

```
List<T>.Add()
```



Inserting

Insert in middle of list

```
List<T>.Insert()
```

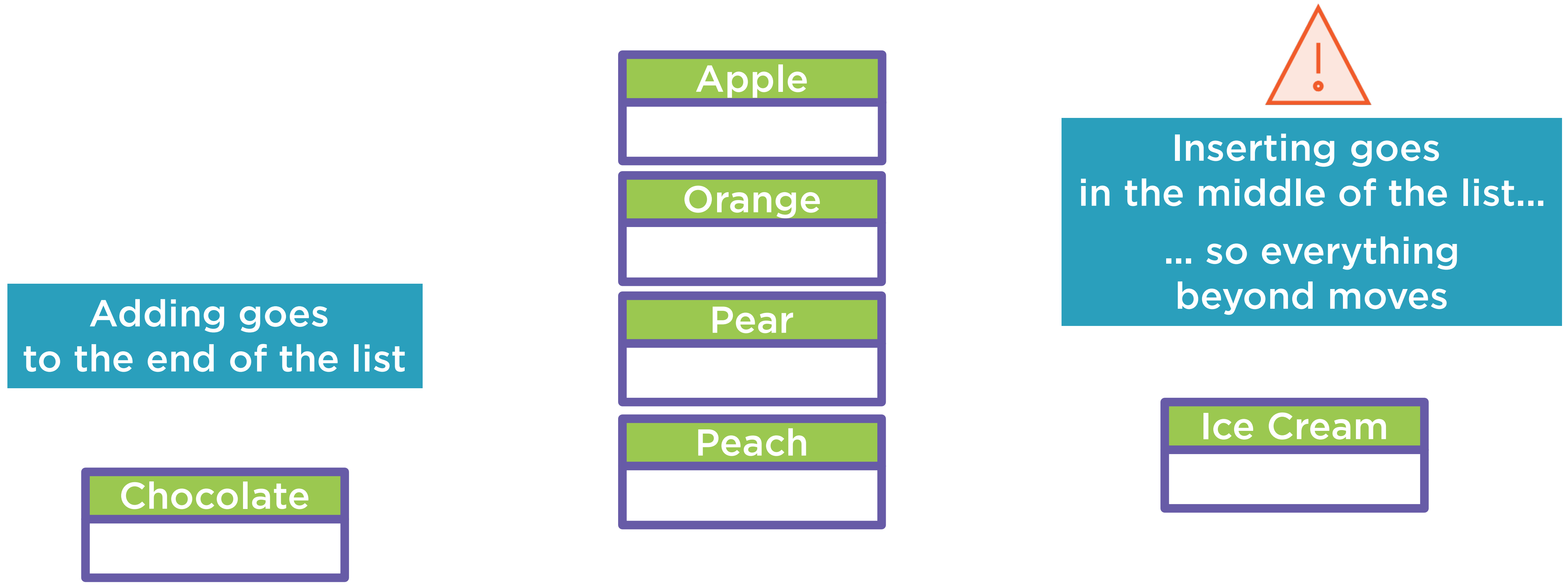
Code Demo

Performance

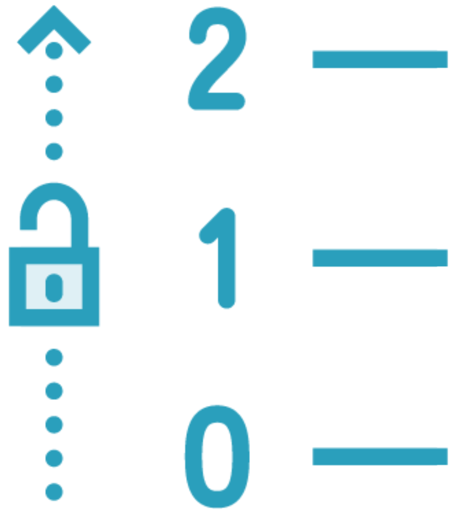
Adding goes
to the end of the list

Inserting goes
in the middle of the list…

… so everything
beyond moves



Inserting and Removing



Data really does move

- Fine for small lists
- Be careful of big lists

Same for inserting and removing

Prefer to add where possible

Summary

List<T>

- Starts empty, then add values
- Enumerate/lookup just like arrays
- List.Count, Array.Length
- Search with FindIndex
- Insert/Delete can be inefficient

Manipulating List Data

Simon Robinson
LEAD SOFTWARE DEVELOPER

@TechieSimon   www.SimonRobinson.com

Overview

for **loop**

- Batching data
- Changing enumeration order
- Modifying a collection
- Keeping indices in sync

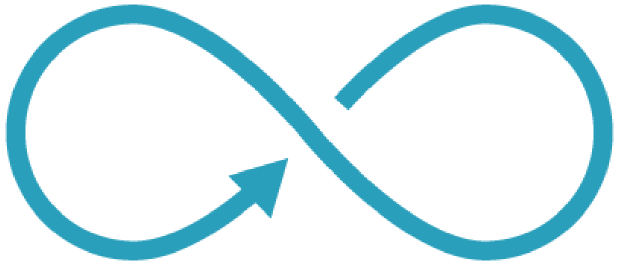
foreach Loop

Very simple

Standard way of enumerating

No control

for Loop



Lower level technique

More control

But more complex code

Can do tasks impossible with foreach

Demo

From ReadAllCountries demo

- Convert foreach to for
- To understand how for works

Code Demo

From earlier in the course...

```
Country[] countries = reader.ReadFirstNCountries(10);

foreach (Country country in countries)
{
   // etc
```

This enumerated the first 10 countries

- But by only importing 10 countries
- So there were only 10 countries in the array

From earlier in the course...

```
for (int i = 0; i < nCountries; i++)
{
    string csvLine = sr.ReadLine();
    countries[i] = ReadCountryFromCsvLine(csvLine);
}
```

for loop to import the countries

Code Demo

Demo

Option to view more countries

- Batch the countries
- (Or: Pause the iteration)
- Easy with for loop

Code Demo

Demo

Display position of each country

- Display ‘1’ for 1st country etc.
- Requires a for loop

Code Demo

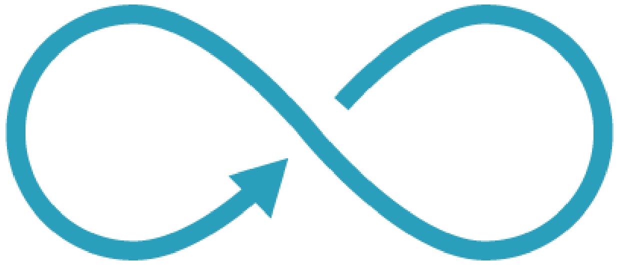
Demo

Display countries in reverse order

- Count down from last item in list

Code Demo

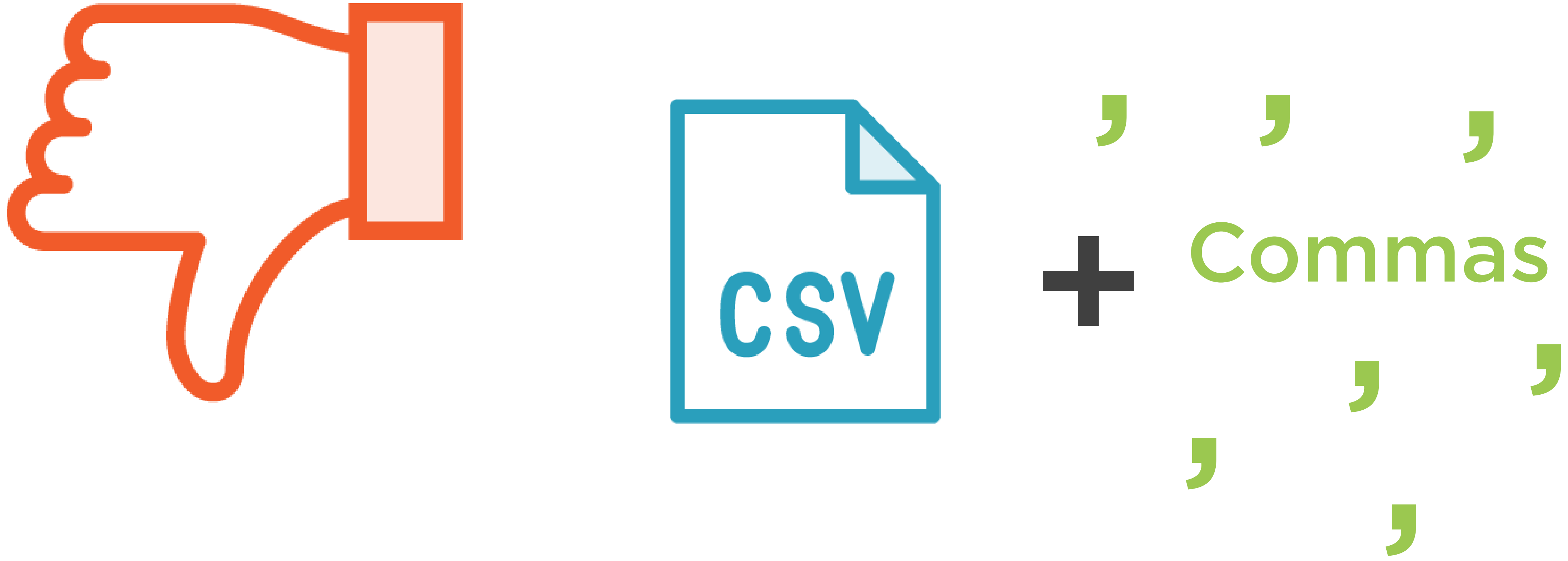
for Loop



Considerable control when enumerating

How about modifying?

- For example, removing items


CSV
+
Commas

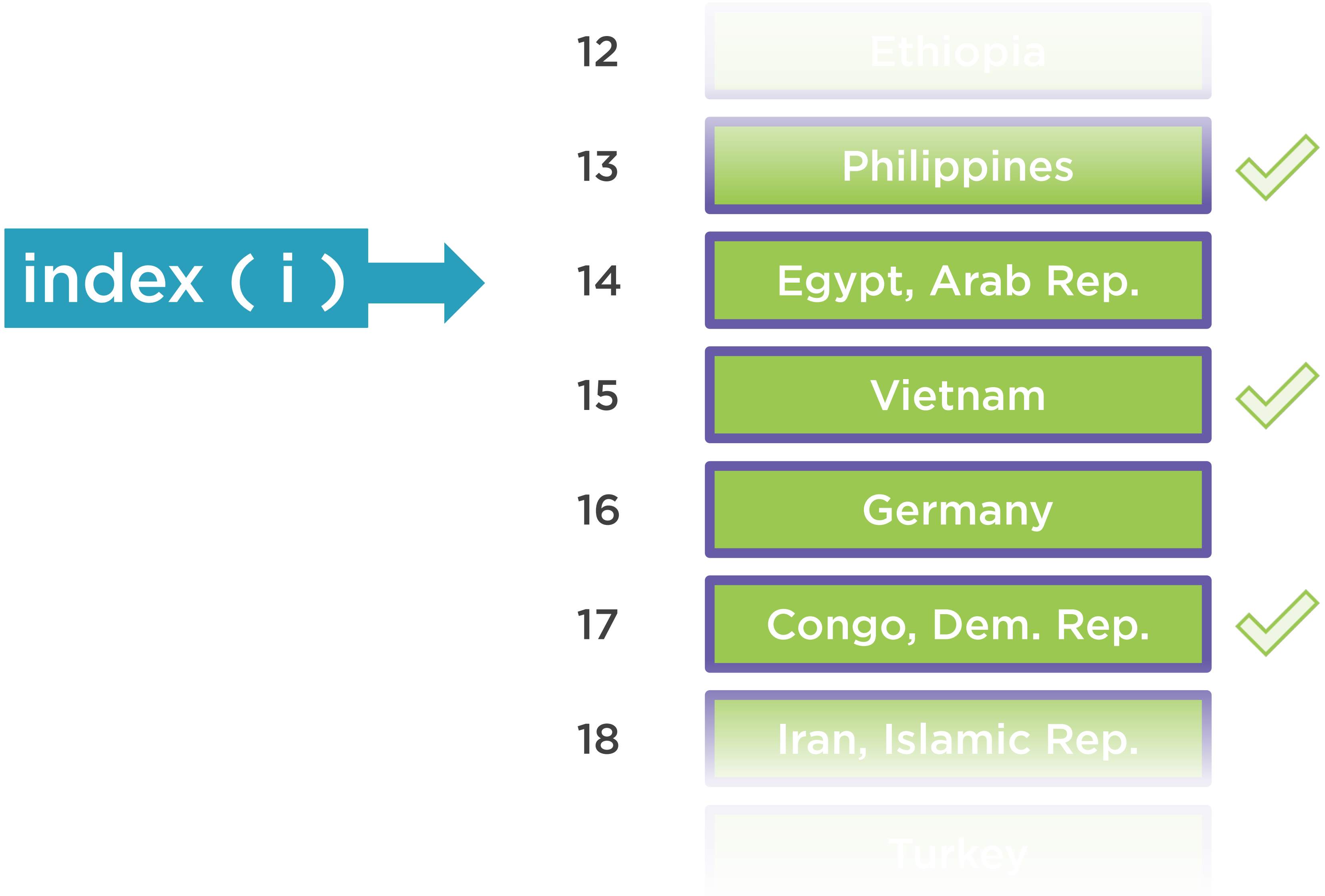
Demo

Modifying the list

- Remove countries with commas

Code Demo

Removing Countries



Two Solutions

Avoid incrementing counter after delete

Complicated

Work backwards

This just works

Code Demo

foreach is only for reading a collection

- Use **for**

to modify a collection

Code Demo

Summary

Arrays and Lists

- `for` loops give access to the index
  - Batch items
  - Change enumeration order
- `for` loops let you modify collections
  - but work backwards
- `List<T>.RemoveAll()`

Storing Keyed Data with Dictionaries

Simon Robinson
LEAD SOFTWARE DEVELOPER

@TechieSimon   www.SimonRobinson.com

Overview

Dictionaries

- Look up items with a key
- Great for unordered data

Coding tasks

- Enumerating
- Adding/removing
- Looking-up

The Demo Code

Demo code will allow choosing a country:

User types in:	A country code
App displays:	Info about that country

The Demo Code

Demo code will allow choosing a country:

User types in:	FRA
App displays:	France
	(+ information about France)

The Problem

We need:

$$\begin{tikzcd}\text{Country code} \arrow[r] & \text{Country}\end{tikzcd}$$

Array and List give:

$$\begin{tikzcd}\text{Index in collection} \arrow[r] & \text{Country}\end{tikzcd}$$

Can You Search?

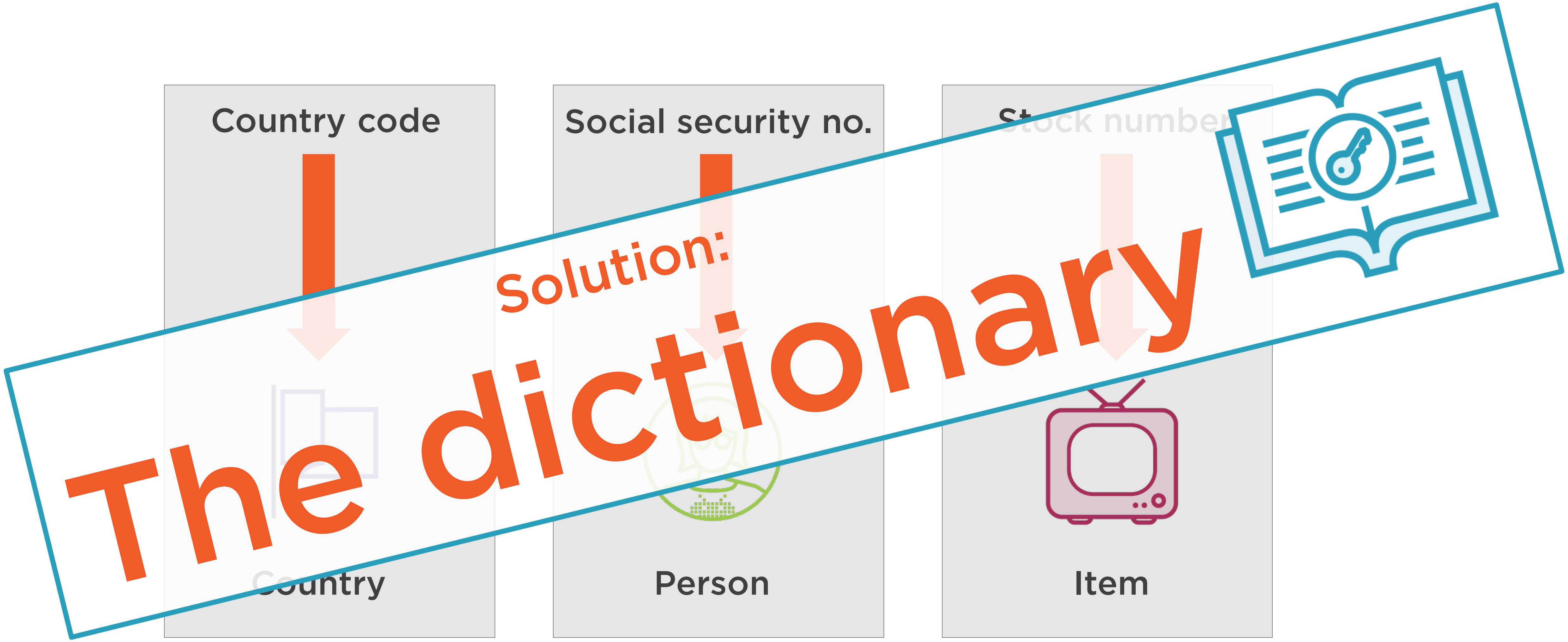
Would this work...?

```
int index = countries.FindIndex(x => x.Code == "FRA");
Country selectedCountry = countries[index];
```

Yes...

- ✗ **... but it's complicated**
- ✗ **... and inefficient**

Looking Items up (Without an Index)



Dictionaries

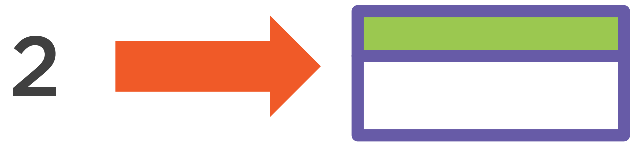
- totally different from arrays and lists

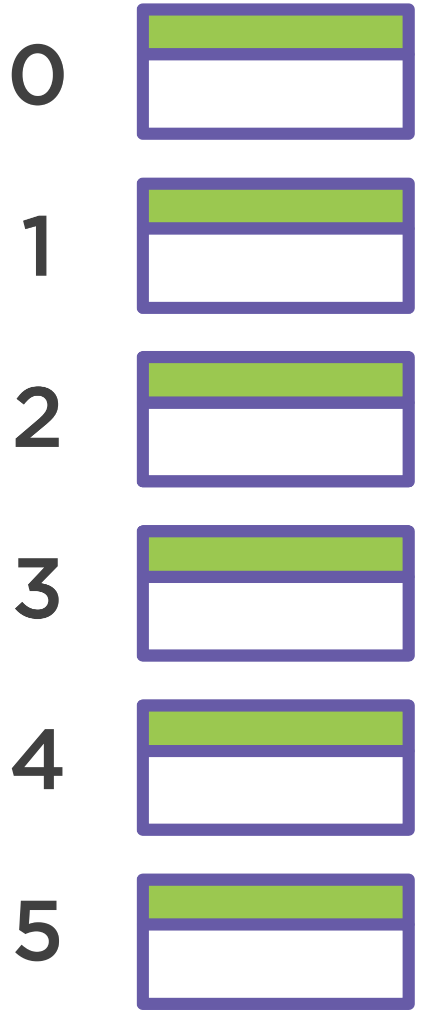
Dictionary vs. Array / List

Array and List

Items in order

Look up using index



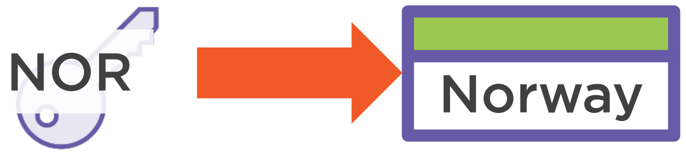


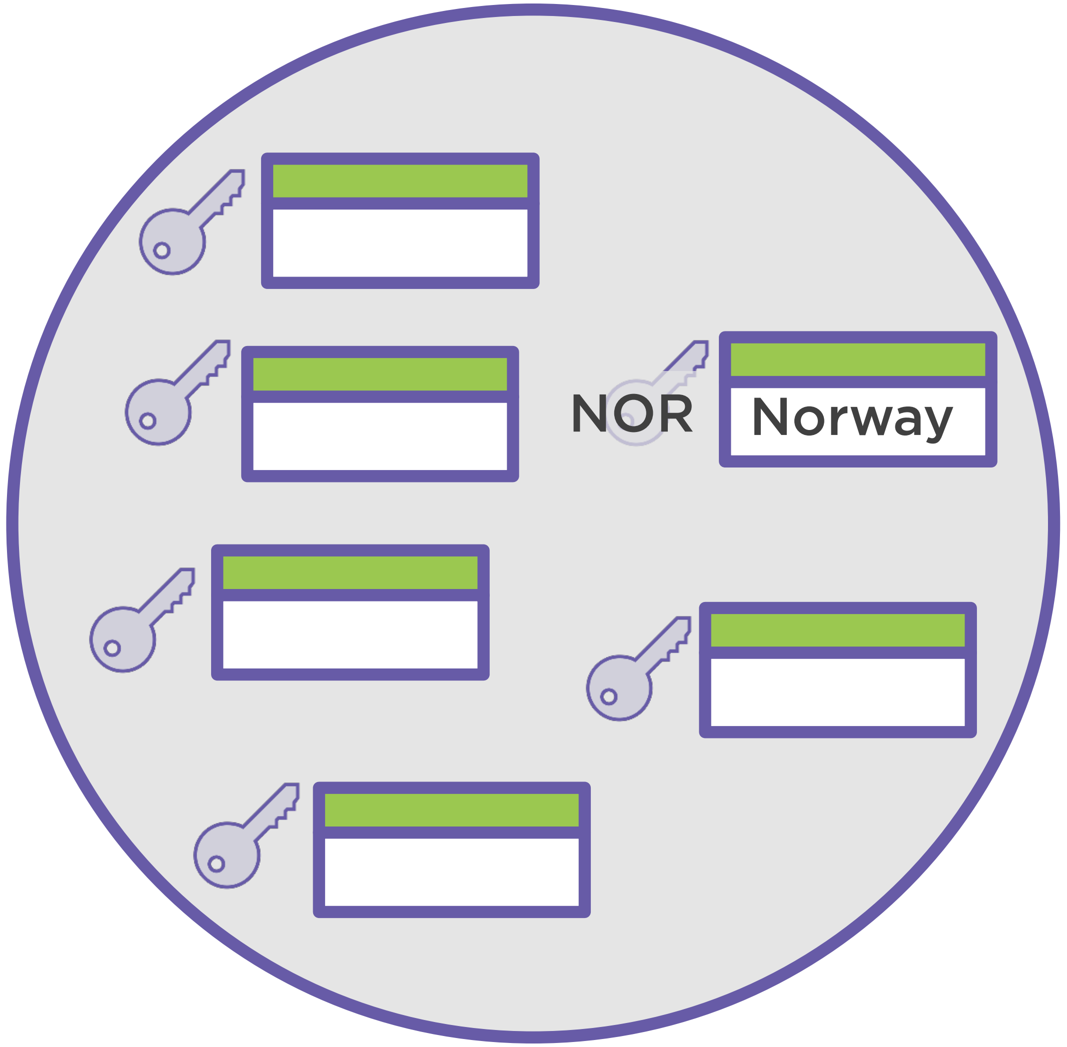
Dictionary vs. Array / List

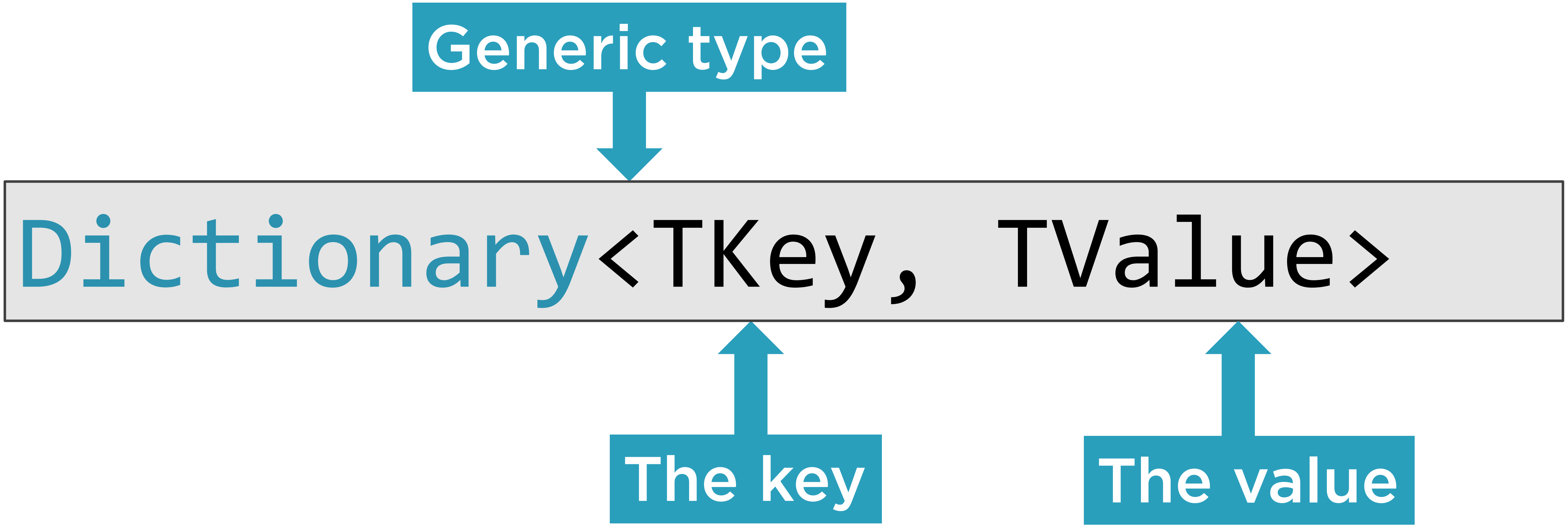
Dictionary

Think of as 'random bag'

Key gives access to the item







```
Dictionary<TKey, TValue>
```

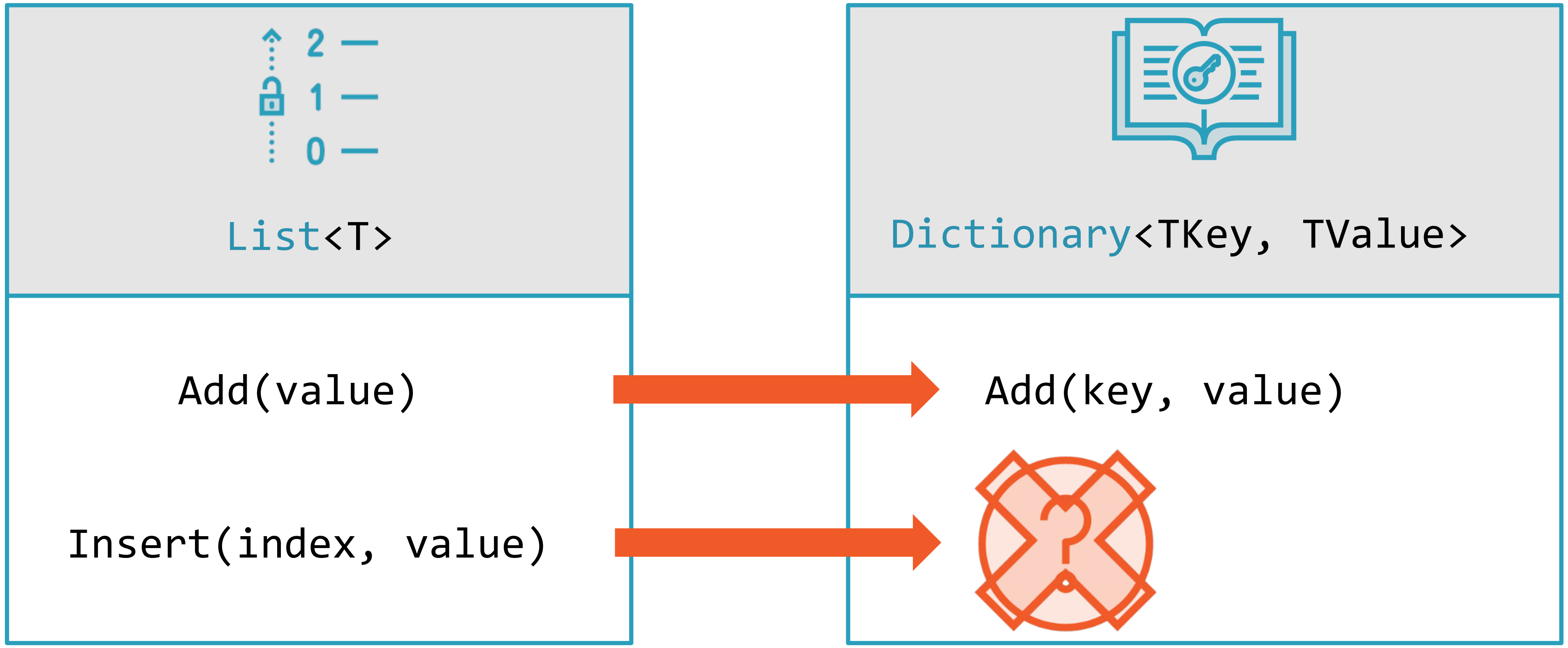
Demo

Basic dictionary coding

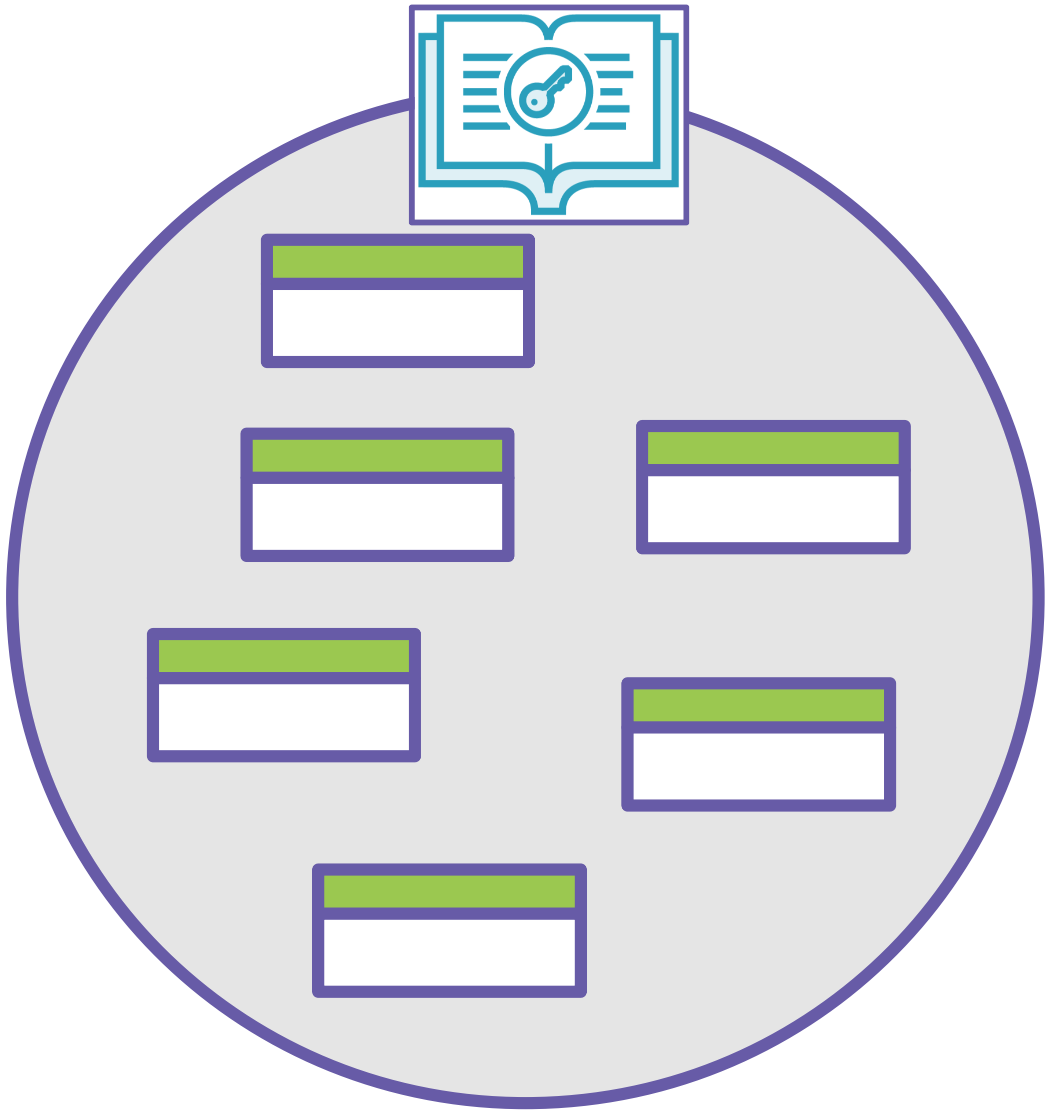
- Looking up
- Enumerating

Code Demo

List vs. Dictionary



Dictionaries Are Not Ordered



Inserting makes no sense

Because items not ordered

Code Demo

Square Bracket Syntax

All collection types:

```
Country country = countries[...];
```

Array and List:

```
country = countries[index];
```

Index is an integer

Dictionary:

```
country = countries[key];
```

Key can be any type

Square Bracket Syntax

All collection types:

```
Country country = countries[...];
```

[] = look up an item

For most collection types

Syntax for collection operations is (roughly) the same

- no matter what the collection type

Code Demo

Comparing Collections

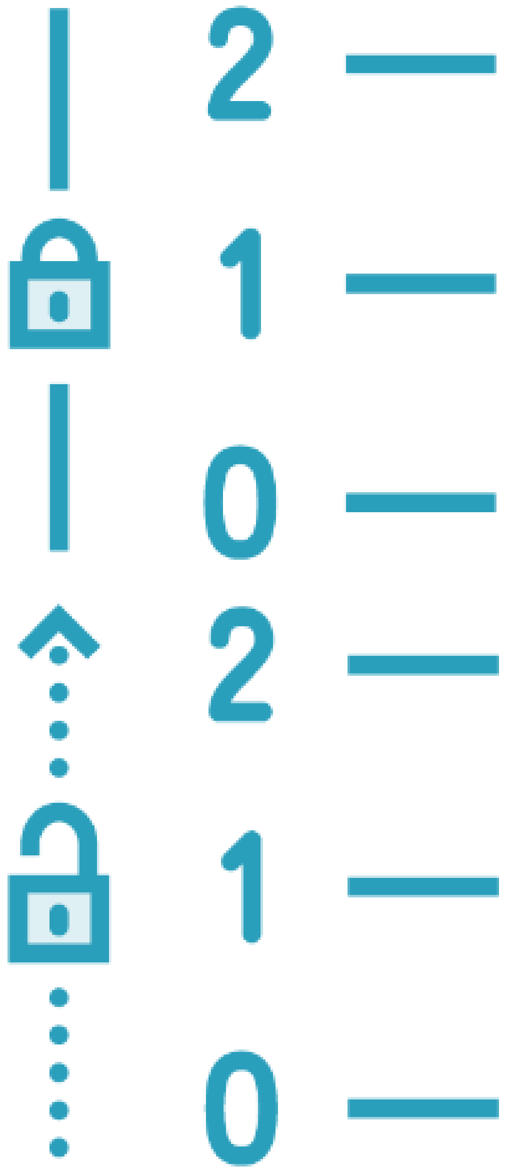
Dictionary operations

- Similar syntax to arrays and lists
- But implementations differ
  - E.g. requiring keys

Demo

Dictionary initializers

Code Demo



Arrays and Lists

Easy to tell what indices are valid:

- Valid indices are `0` to no. of items `- 1`
- `Array.Length`, `List.Count` tell you no. of items

Code Demo

Other Dictionary Features

`Remove()` **method**

`[]` **for replacing items**

- Item specified with Key
- Beware of whether key exists

`ContainsKey` **property**

Code Demo

Demo

Countries

- Allow user to choose a country using country code
- Requires a dictionary keyed by country code

Code Demo

Summary

Dictionaries

- Great for direct look-up
- Accessed by key, no order required
- Look-up/enumerating: Same syntax as lists and arrays
- But beware implementation differences

Next up: Techniques for accessing collection data

Taking Collections Further

Simon Robinson
LEAD SOFTWARE DEVELOPER

@TechieSimon www.SimonRobinson.com

Overview

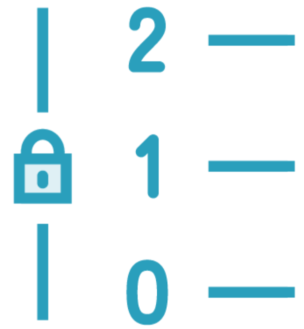
Recognise when you need more advanced techniques

Taking collections further

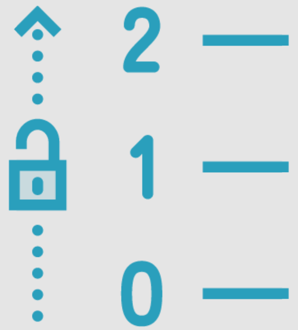
- More generic collections
- Immutable collections
- Concurrent collections
- LINQ for other data sources
- Interfaces

Other Pluralsight courses

Recap



`T[]`



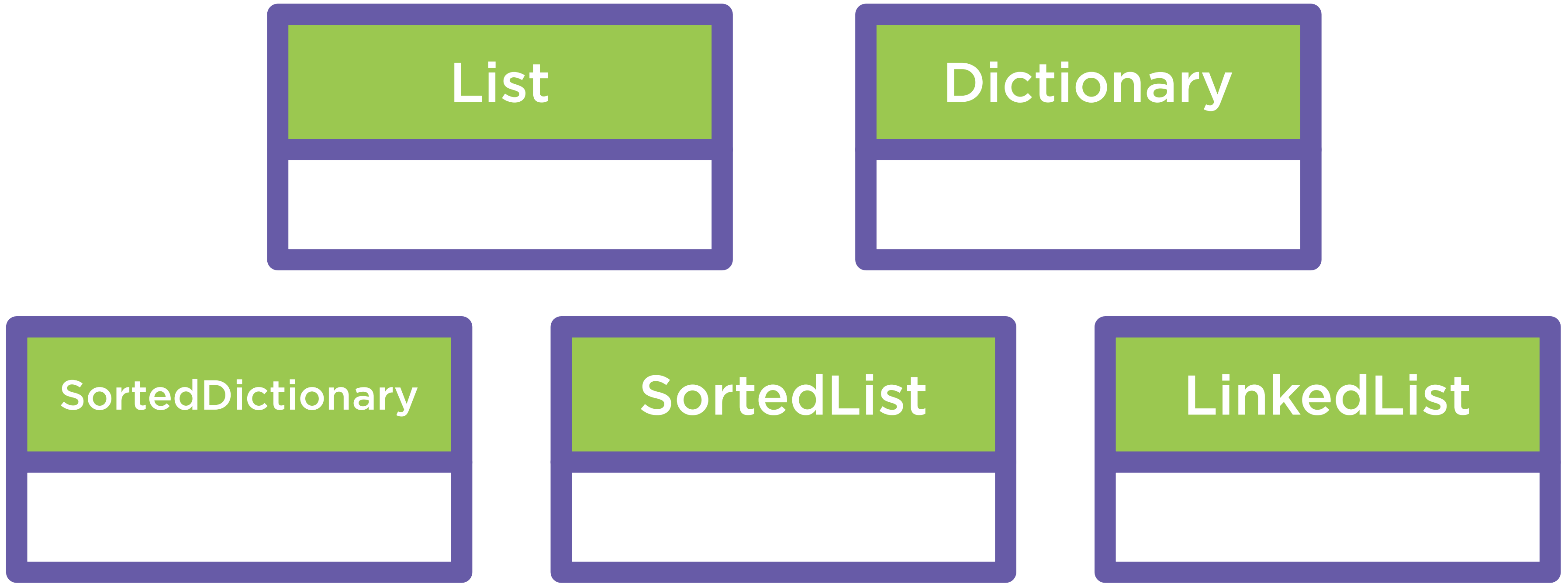
`List<T>`

`Dictionary<TKey, TValue>`

Standard generic collections

Standard Generic Collections

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

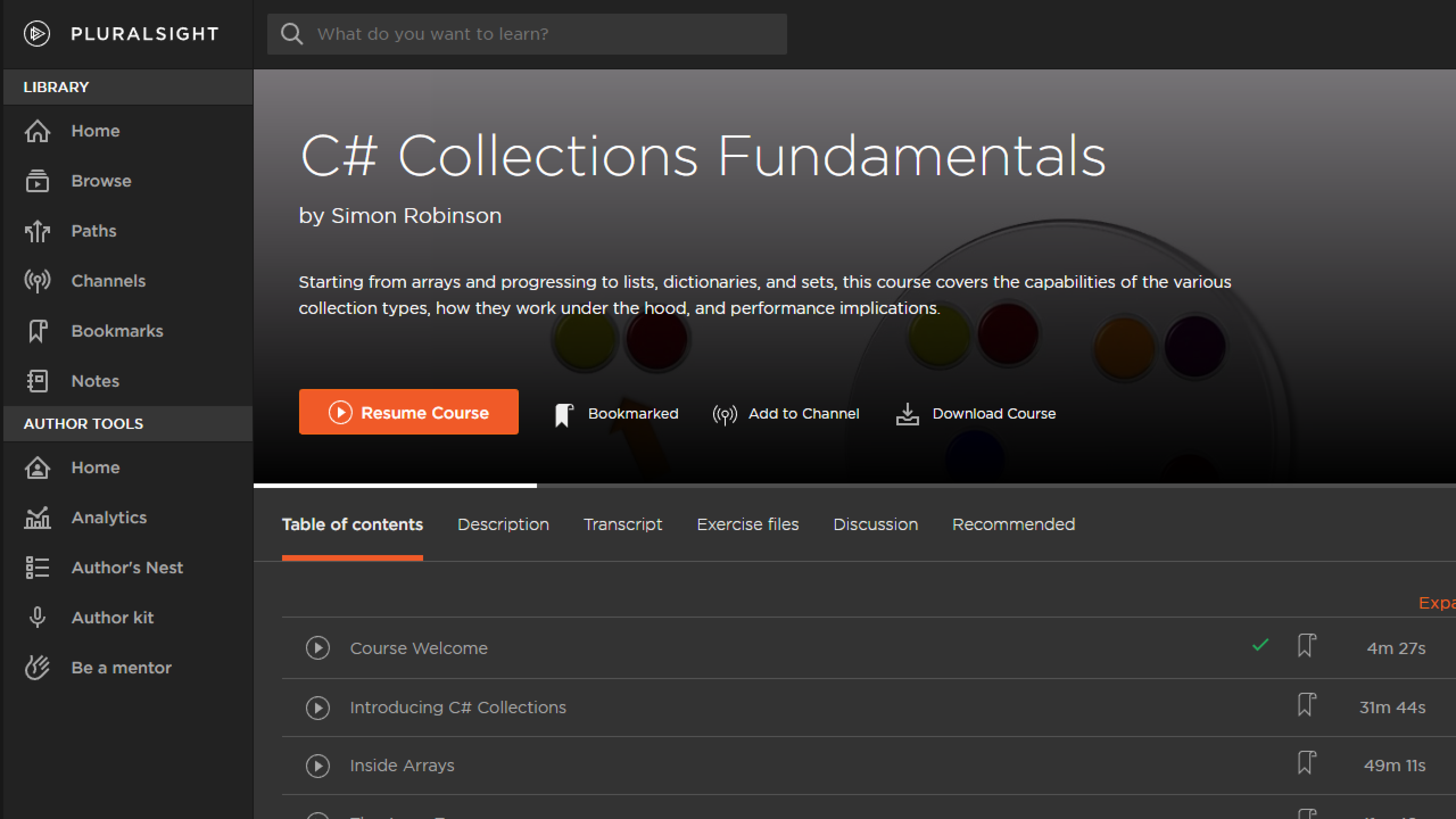


Build-your-own Collections

```
using System.Collections.ObjectModel;
```

ObservableCollection

- Notifies when something changes
- Built using ObjectModel types
- You can do the same thing



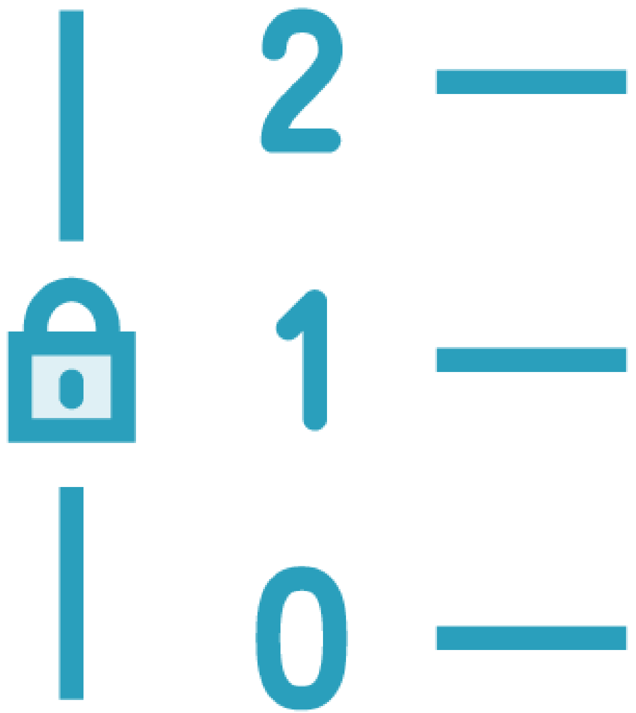
C# Collections Fundamentals

by Simon Robinson

Starting from arrays and progressing to lists, dictionaries, and sets, this course covers the capabilities of the various collection types, how they work under the hood, and performance implications.

Course Welcome ✓ 4m 27s
Introducing C# Collections 31m 44s
Inside Arrays 49m 11s

Array



Not a standard generic collection

- Uniquely baked into .NET runtime
- Special syntax

Immutable Collections

Immutable

Cannot ever be modified, once instantiated

Immutable Collections

Standard	Immutable
Array	ImmutableArray
List	ImmutableList
Dictionary	ImmutableDictionary

Robust code

Thread safety

Concurrent Collections

Similar to standard collections…

… but thread-safe

Thread Safety

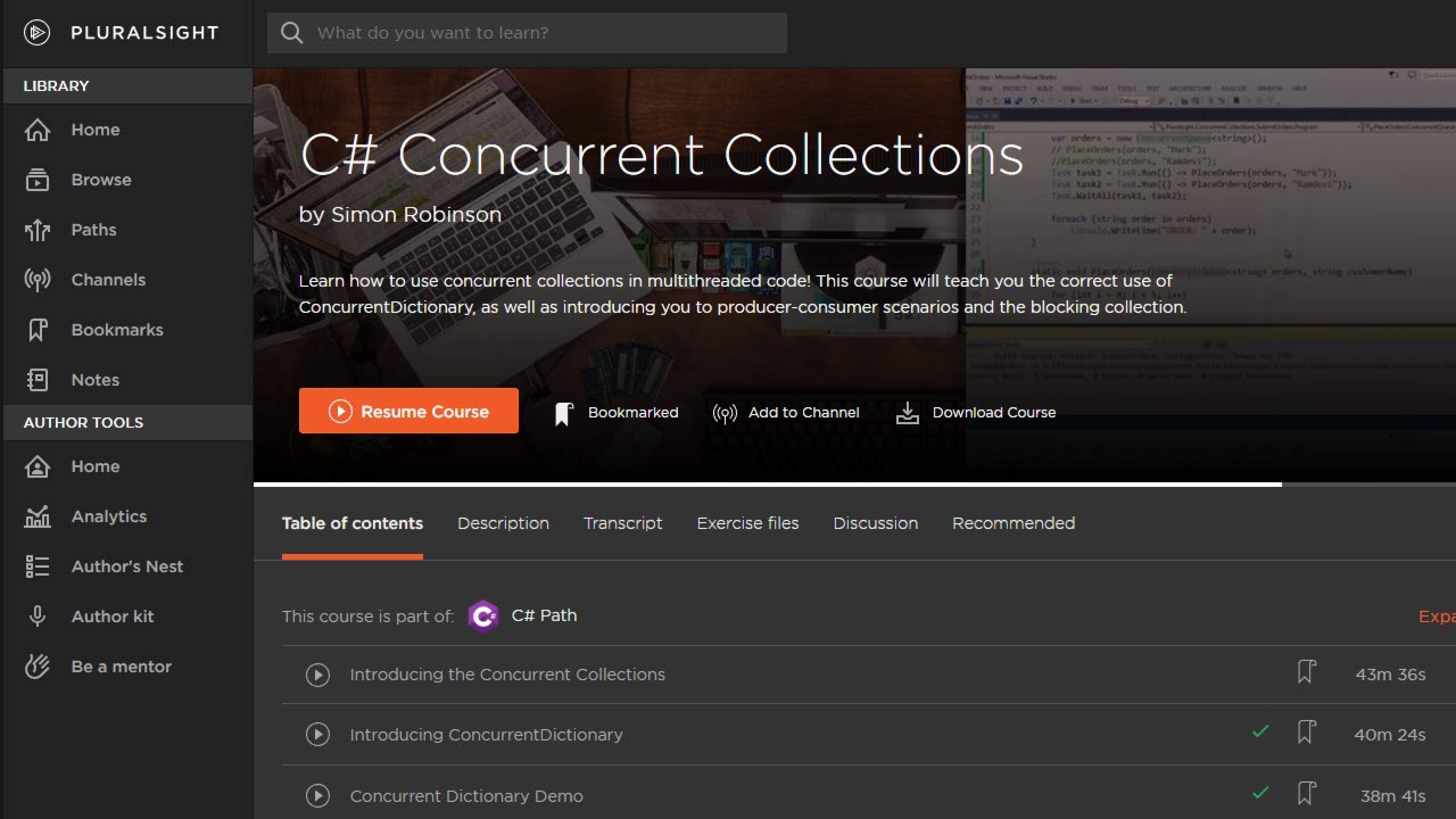
✓

Concurrent collections

Immutable collections

✗

Standard collections



C# Concurrent Collections

by Simon Robinson

Learn how to use concurrent collections in multithreaded code! This course will teach you the correct use of ConcurrentDictionary, as well as introducing you to producer-consumer scenarios and the blocking collection.

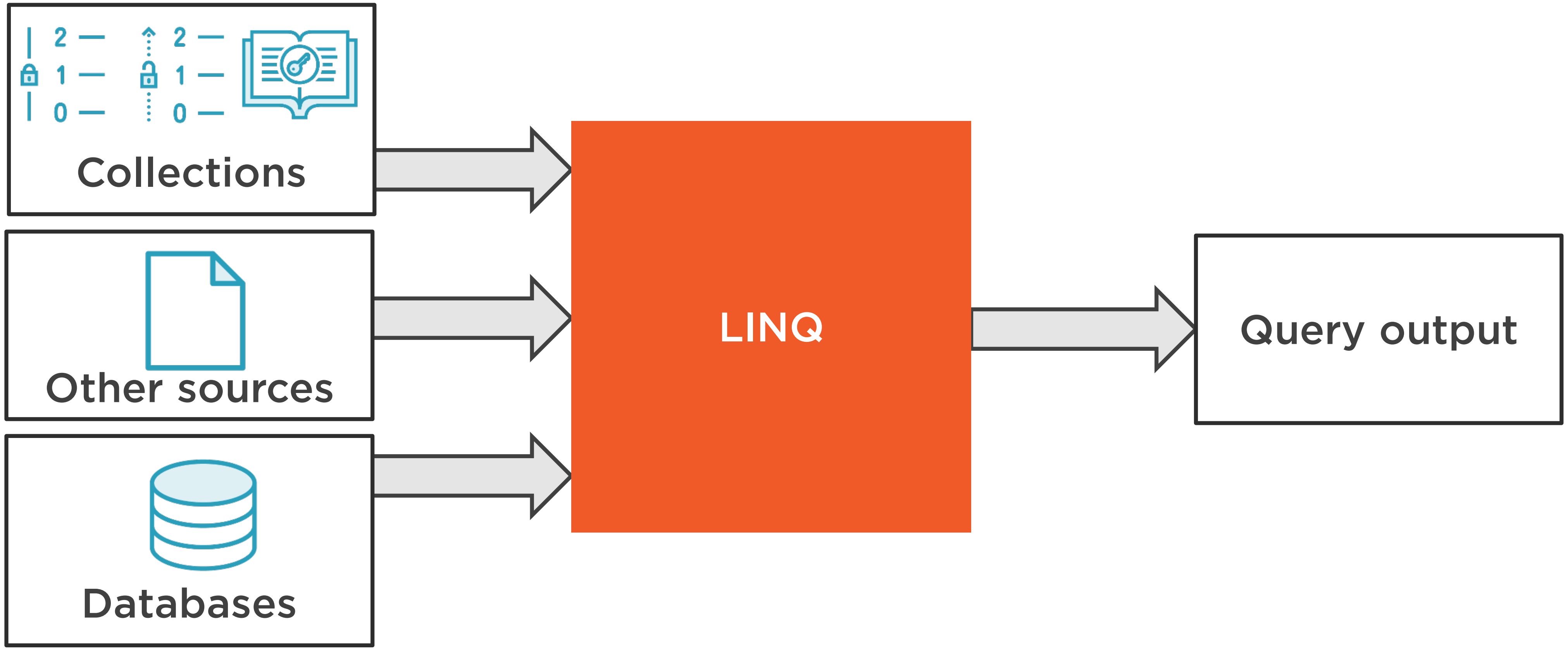
This course is part of: C# Path

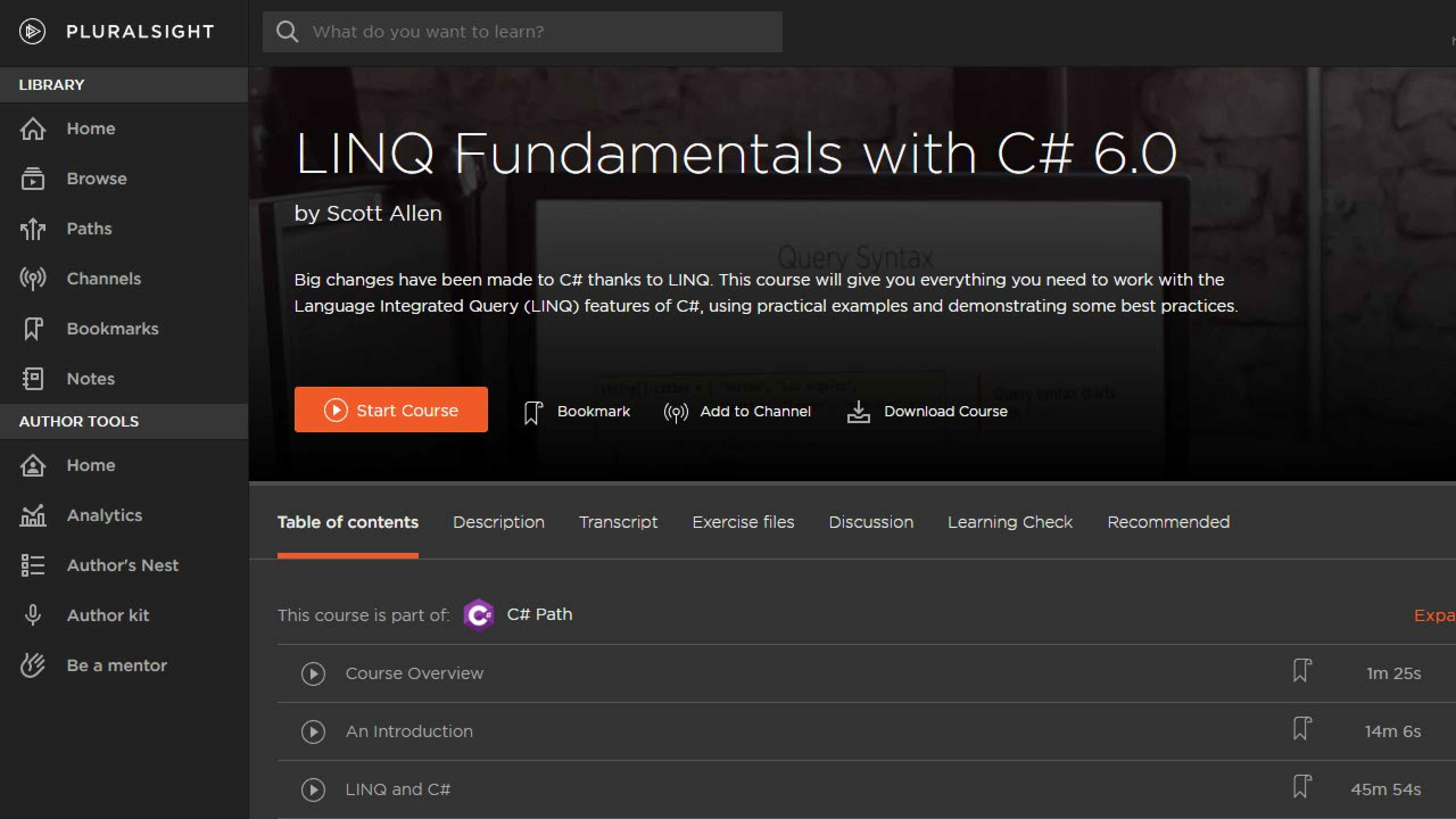
Expa

Introducing the Concurrent Collections	43m 36s
Introducing ConcurrentDictionary ✓	40m 24s
Concurrent Dictionary Demo ✓	38m 41s

LINQ

Complete framework for querying data





PLURALSIGHT

What do you want to learn?

LIBRARY

Home

Browse

Paths

Channels

Bookmarks

Notes

AUTHOR TOOLS

Home

Analytics

Author's Nest

Author kit

Be a mentor

LINQ Fundamentals with C# 6.0

by Scott Allen

Big changes have been made to C# thanks to LINQ. This course will give you everything you need to work with the Language Integrated Query (LINQ) features of C#, using practical examples and demonstrating some best practices.

Start Course

Bookmark

Add to Channel

Download Course

Table of contents

Description

Transcript

Exercise files

Discussion

Learning Check

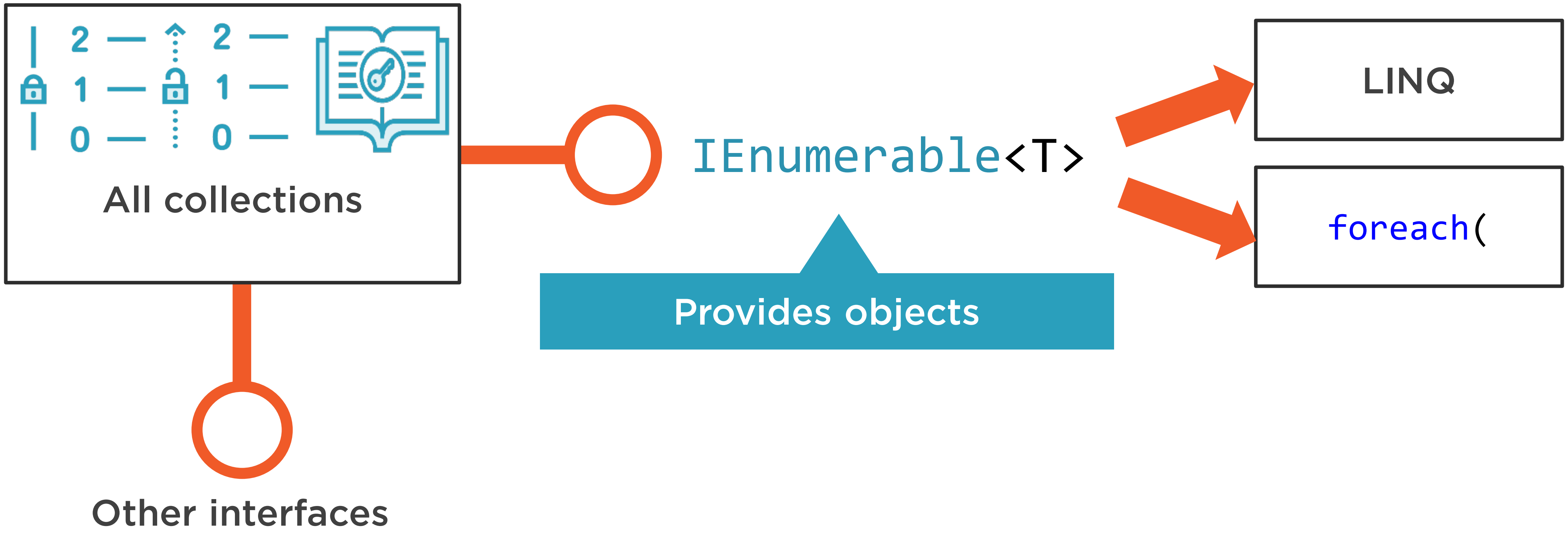
Recommended

This course is part of: C# Path

Expa

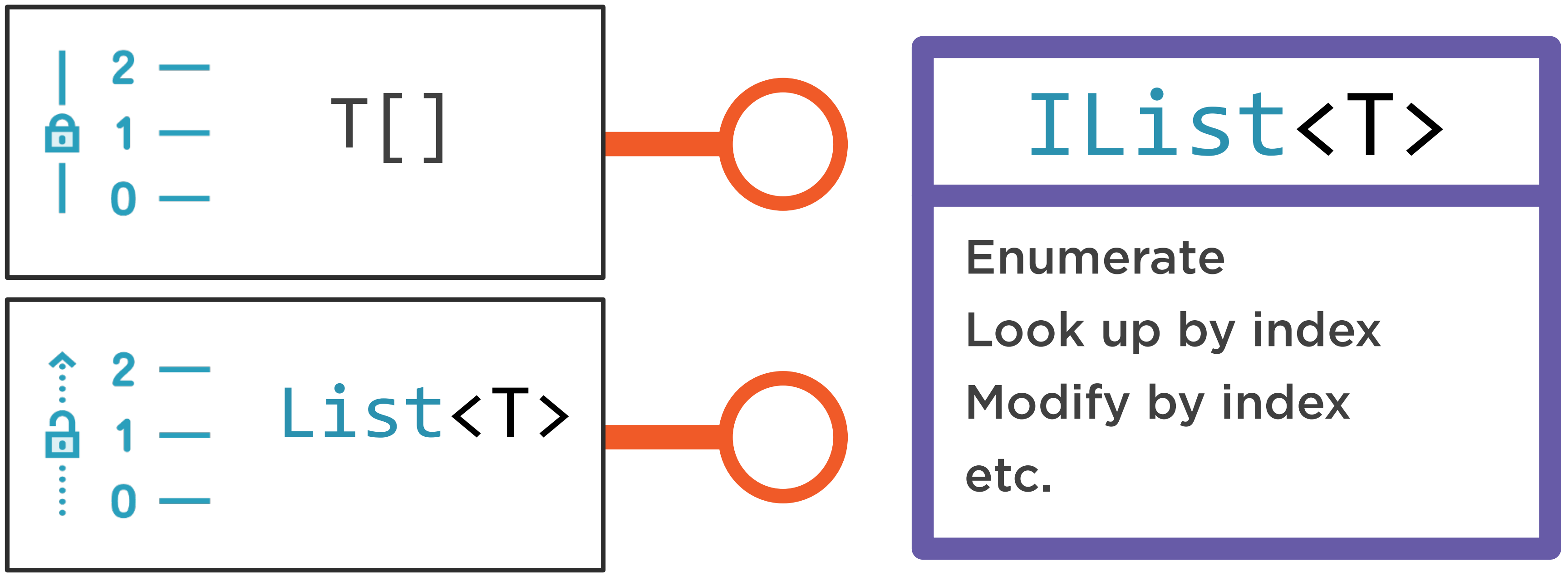
Course Overview	1m 25s
An Introduction	14m 6s
LINQ and C#	45m 54s

Interfaces



All collections
IEnumerable<T>
Provides objects
LINQ
foreach(
Other interfaces

Interfaces



Enumerate

Look up by index

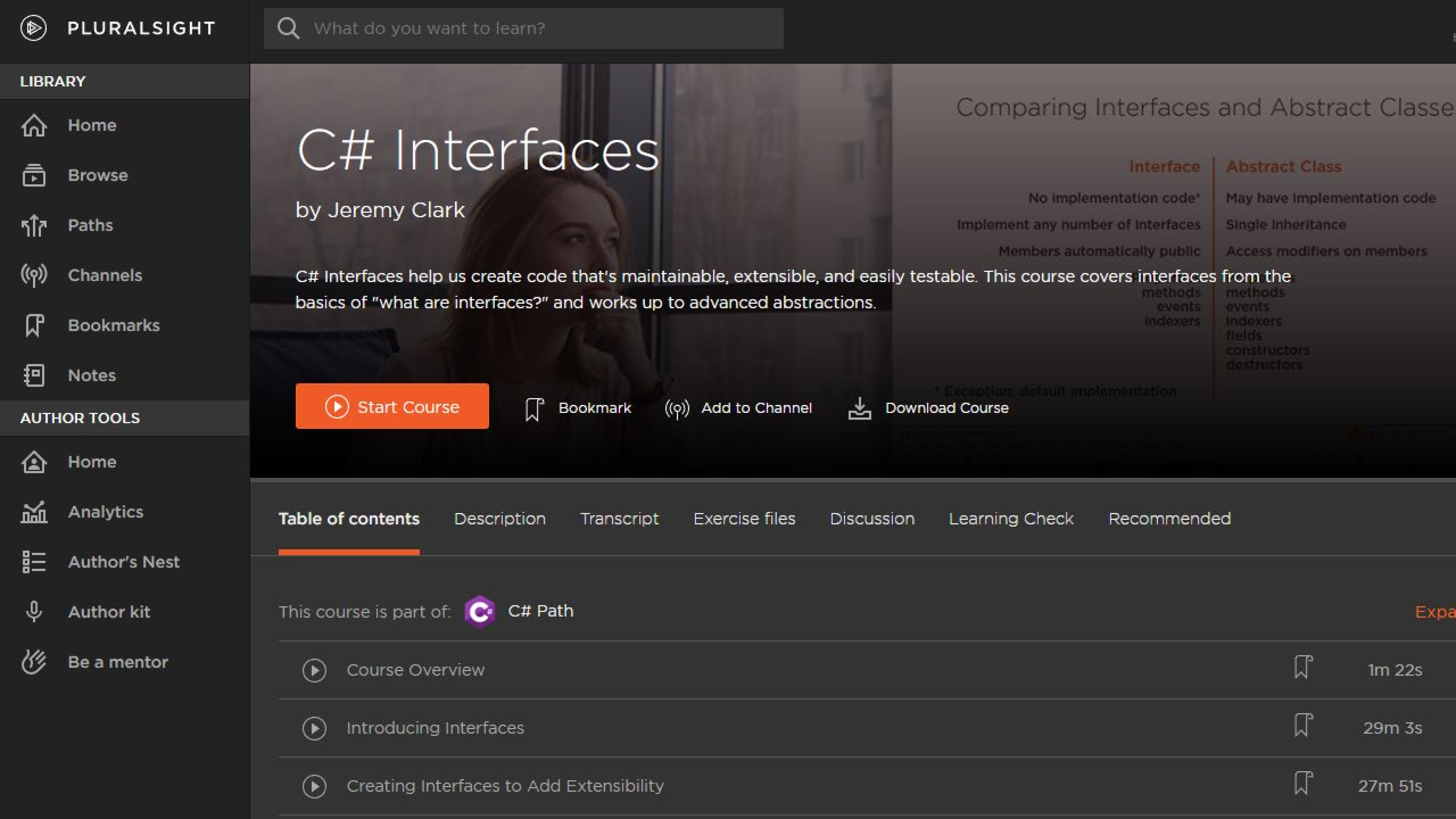
Modify by index

etc.

Demo

Consume a list using IList<T>

CODE DEMO



Comparing Interfaces and Abstract Classe

Interface	Abstract Class
No implementation code*	May have implementation code
Implement any number of interfaces	Single inheritance
Members automatically public	Access modifiers on members
methods	methods
events	events
indexers	indexers
	fields
	constructors
	destructors

* Exception: default implementation

C# Interfaces

by Jeremy Clark

C# Interfaces help us create code that's maintainable, extensible, and easily testable. This course covers interfaces from the basics of "what are interfaces?" and works up to advanced abstractions.

This course is part of: C# Path

Course Overview	1m 22s
Introducing Interfaces	29m 3s
Creating Interfaces to Add Extensibility	27m 51s

Course Summary

Arrays and lists for ordered data

Dictionaries for direct (keyed) access

Accessing elements
- foreach loop
- for loop
- LINQ

Collections of collections