Lessons 1-2

Data Types
==========
Comments: // Single line ... /* Multiline */
string => alphanumeric (defaults to null)
int => 2 billion to -2 billion (defaults to 0)
double => fractional value (defaults to 0.0)
bool => true or false (defaults to false)

Data Type Conversion
==========
Implicit conversions - smaller type to larger type without data loss, "upcasting"
Explicit conversions - require developer intervention, possibility of data loss, "downcasting", either in the form of cast or using a helper method.

Casting numbers: int myInteger = (int)myDouble;

Numbers to strings: string myString = myInteger.ToString();

String to Numbers: int myInteger = int.Parse(myString);

Lessons 3-4

Arithmetic Operators
==========
= NOT equality, it's assignment
Math Operators: + - * /
Addition Assignment
total = total + 5;
total += 5;

Increment Operator: i++;
Decrement Operator: i--;

Beware of order of precedence (use parenthesis)
Beware of down casting (you'll lose precision)
Beware of overflow (use bigger types)

To make overflow throw an exception:

```
checked
{
// some arithmetic operation
// that could potentially overflow
}
```

Lessons 5-6

C# Syntax
==========
Operands - variable names, object / server control names, literals - "Nouns" (you name these)

Operators - "Verbs ... they act on the operands.

Expressions - One or more operands and zero or more operators that evaluate to a single value

Statements - A complete instruction - assignment of an expression to a variable, an increment/ decrement, etc.

Statements must end in a semi-colon ;
Whitespace is ignored (use for humans)

Conditional if ... else if ... else Statement
==========
= Assignment
== Equality

```
if (a == b)
{
// execute when the expression is true
}
else
{
// executes when the expression is false
}
```
... or ... evaluate other mutually exclusively options:

```
if (a == b) { // some code }
else if (a == c) { // some code }
else if (a == d) { // some code }
else { // catch all }
```

CheckBox Server Control = Checked prop is bool

RadioButton Server Control = GroupName prop groups them together, check prop is bool

Lessons 7-8

Conditional Ternary Operator
==========
Shortcut for evaluating an expression and returning a result.
result = (a == b) ? "Equal" : "Not Equal";

Lessons 9-10

Comparison and Logical Operator
==========
Comparison Operators used for conditional statements
==
!=
< >
<= >=
!someBooleanValue - means NOT is true

Logical Operators
used to combine multiple expressions / evaluation

&& - AND
|| - OR

Combine with parenthesis ( ) for order of precedence

Single Dimensional Arrays
==========
Indexes vs. Elements
Accessor vs. Stored Values
Indexes are zero based

Declaring Arrays
string[] myArray = new string[3];

Declaring and Initializing Arrays
string[] myArray = new string[3] { "Moe", "Larry", "Curly" };

Setting / Getting Values
string myString = myArray[1]; // Retrieve the second element
myArray[0] = myString; // Sets first element

Multi-Dimensional Arrays
==========
Same as single dimensional ... just requires more indexes (in dimensions) to get to the element

```
double[,] myArray = new double[2,3]; // contains 6 elements
int[,,] rubicsCube = new int[3,3,3] // contains 27 elements

rubicsCube[0,1,2] = 42;
myInteger = rubicsCube[0,1,2];
```

Changing the Length of an Array
==========
Arrays are immutable = cannot be changed in memory
HOWEVER, .NET Framework providers helper methods to resize an array ... creates a new array and copies the old values into it.

```
Array.Resize(ref myArray, myArray.Length + 1);

// Get the highest index:
int highestIndex = myArray.GetUpperBound(0);
// 0 = dimension we want to retrieve the
// upper boundary for

// Arrays have other helper methods
myArray.Sum()
myArray.Min()
myArray.Max()
myArray.Average()

Array.Sort(myArray)
Array.Reverse(myArray)
```

Lessons 11-12

Creating a Database in Visual Studio

LocalDb - Local dev-only version of SQL Server

Project > Add New Item ... > Installed > Data > SQL Server Database

Creates an .mdf file

SQL Server Data Tools (SSDT) - Tools to create and manage SQL Server

databases from Visual Studio.

Creating an Entity Model in Visual Studio

Entity Data Model - Object-Relational Mapper to treat database tables w/

columns as classes w/ properties

Project > Add New Item ... > Installed > Data > ADO.NET Entity Data Model >

Entity Data Model Wizard > EF Designer from Database > Connection > Database

Objects

DbContext == Handle to the entity model > database
DbSet == Collection of all entities in the DbContext

```
ACMEEntities db = new ACMEEntities();
var dbCustomers = db.Customers;
```

Displaying the DbSet Result in an ASP.NET GridView

GridView Server Control - Databinds to enumerable collections of objects and

renders in a tabular format

Must call ToList() on a DbSet to bind to a databound control.

```
gridControl.DataSource = dbCustomers.ToList();
gridControl.DataBind();
```

Lessons 13-14

Implementing a Button Command in a GridView
==========
Click Chevron => GridView Tasks > Edit Columns...

BoundField - Databind to a object property
ButtonField - Hyperlink button

Handle button click in the GridView_RowCommand event handler.

protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs

e)
{
// Retrieve the ROW CLICKED in the grid
int index = Convert.ToInt32(e.CommandArgument);
GridViewRow row = GridView1.Rows[index];

// Accessing cells is risky because the order
// of the columns may change over time
// (and you might forget that this code
// depends on it!)
// Also ... 0 based!
var someValue= row.Cells[1].Text;

}

Using a Tools-Centric Approach to Building a Database Application
==========
Tools-Centric approach / workflow = Use Visual Sudio's designers, tools, etc. to build applications w/ minimal code.

Great for small, departmental apps with very little business logic, change is not anticipated and there's a tight timeframe.

Lessons 15-16

Using a Maintenance-Centric Approach to Building a Database Application
==========

Maintenance-Centric approach / workflow = Anticipate change, mitigate it's

negative effects on software by separating concerns, applying unit testing,

etc.

Great for larger, enterprise scale apps with many business rules, where

change is anticipated because it is crucial to the operation of the business

and there's a longer development timeframe.

DTO - Data Transfer Object -- model used for transfering
from one layer to another to avoid a leaky abstraction.
Ex., I don't want Entity Framework leaking out of
persistence because other layers would be dependent on
it!

Lessons 17-18

Creating a New Instance of an Entity and Persisting to the
==========

```
var customer = new Customer();
// Populate properties of customer
dbCustomers.Add(customer);
db.SaveChanges();
```

Filtering DbSets using LINQ method syntax:

```
ACMEEntities db = new ACMEEntities();
var dbCustomers = db.Customers.OrderBy(p => p.Name).ToList();

.Where(p => p.Name == "Bob").ToList();
```

Lambda Expression - "mini methods"