***5/21/2018 (Day 1):***

*For jobs: I was a proofreader for 2 years and one of the best they had…I will be able to read through code and find syntax and logic errors better than most people.*

**GIT & GITHUB**

Git and github are 2 technologies that perform one function

***GIT*** = source control management(SCM) / backs up to cloud

Push code to repository (cloud) at least 1x per day

***REPO*** = repository

Keep repositories organized

Store all REPOs in c:\repos

DO NOT DO A GIT INIT IN REPOS!!!

~ = sitting in your home directory

Create all of our code inside c:\repos with subfolders



!!!NOTE that /c/repos is not a repository!!! --- The real repositories will be in subfolders

***SOURCE CONTROL*** = manages multiple versions of documents and keeps it in order

***GITHUB*** = same thing as GIT but in the cloud

Topics for GIT AND GITHUB today include: (1) Bash shell (see image above); (2) Command line interface (CLI); (3) File structure; and (4) Commands

***Bash shell:***

Similar to Linux command line

Commands are in book (pg.14 or so in book)

Also Greg will be getting us the slides for this presentation

***Command Line Interface (CLI):***

Git commands entered at command line (git cmd [opts}

In book on page 14

git: program name

cmd: what to do

[opts]: options based on command

All commands are lower case

>Git status –s

***File structure:***

How to tell if it is a file repository

Add –a to see hidden files

If it has a .git folder inside then it is a repository

You don’t want repositories inside of other repositories

Removing the .git folder removes git control

***Commands:***

All commands begin with git (lower case)

See GIT/GITHUB slide printouts for other commands such as init, remote, etc.

If you forget –m on message commits then add comments to top and “:X” in place of insert on text editor that pops up

Steps to add/commit things on github:(slide 17)

1. git init adds folder to start with git
2. git add to start tracking
3. git commit to commit (don’t forget to use -m for message)

***5/22/2018 (Day 2):***

***4 Announcements:***

1. Don’t leave electronics behind or other things behind each night
2. .NET meetup tonight @ 6PM
3. Party tomorrow @ 4:30PM
4. Fill out Day 1 feedback form

***SQL Servers:***

* ***SSMS*** = SQL Server Management Studio (interacts with the SQL table to manipulate it)
* Primary key is a unique identifier (at least one column but can be more than one) when taken together makes the value of that row unique – can use integers and for the bootcamp we will always use “Id” incremented 1 integer at a time (i.e., 1,2,3,4…)
* In SQL strings are bound by single quotes
* To insert a single quote (‘) in a string you must insert two single quotes next to each other (‘’); note that this is not a quotation mark but two single quotes
* Ordering can be different in SQL if your table is in different languages
* can't have any columns in the list that do not have a correspoding "group by" when using a sum
* when grouping data you must either group by or be aggregated or errors will be returned
* "having" works just like where but is used on aggregated data

***GitHub Upload of SQL Lesson:***

|  |
| --- |
| Max-Student@Student03 MINGW64 /c/repos  $ cd sqllesson  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git status  On branch master  Your branch is up to date with 'origin/master'.  Untracked files:  (use "git add <file>..." to include in what will be committed)  SqlLesson Queries/  nothing added to commit but untracked files present (use "git add" to track)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote -v  origin https://github.com/gpdoud/SqlLesson.git (fetch)  origin https://github.com/gpdoud/SqlLesson.git (push)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote remove origin  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote -v  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote add origin https://github.com/michaelbuchanan23/sqllesson.git  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote -v  origin https://github.com/michaelbuchanan23/sqllesson.git (fetch)  origin https://github.com/michaelbuchanan23/sqllesson.git (push)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git status  On branch master  Untracked files:  (use "git add <file>..." to include in what will be committed)  SqlLesson Queries/  nothing added to commit but untracked files present (use "git add" to track)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git add .  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git status  On branch master  Changes to be committed:  (use "git reset HEAD <file>..." to unstage)  new file: SqlLesson Queries/Assignment(TotalByState).sql  new file: SqlLesson Queries/Average.sql  new file: SqlLesson Queries/FromJupiterToSun.sql  new file: SqlLesson Queries/JoinSelectingOnlySpecificColumns.sql  new file: SqlLesson Queries/JoinViews.sql  new file: SqlLesson Queries/Name-City-State.sql  new file: SqlLesson Queries/OuterJoinWithNullValue.sql  new file: SqlLesson Queries/SelectAll.sql  new file: SqlLesson Queries/TotalSalesByCustomer.sql  new file: SqlLesson Queries/Union.sql  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git commit -m "Updated Sql Lesson with Queries from class on 5/22/2018"  [master 41f7d08] Updated Sql Lesson with Queries from class on 5/22/2018  10 files changed, 73 insertions(+)  create mode 100644 SqlLesson Queries/Assignment(TotalByState).sql  create mode 100644 SqlLesson Queries/Average.sql  create mode 100644 SqlLesson Queries/FromJupiterToSun.sql  create mode 100644 SqlLesson Queries/JoinSelectingOnlySpecificColumns.sql  create mode 100644 SqlLesson Queries/JoinViews.sql  create mode 100644 SqlLesson Queries/Name-City-State.sql  create mode 100644 SqlLesson Queries/OuterJoinWithNullValue.sql  create mode 100644 SqlLesson Queries/SelectAll.sql  create mode 100644 SqlLesson Queries/TotalSalesByCustomer.sql  create mode 100644 SqlLesson Queries/Union.sql  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git push origin master  Counting objects: 43, done.  Delta compression using up to 4 threads.  Compressing objects: 100% (39/39), done.  Writing objects: 100% (43/43), 9.06 KiB | 927.00 KiB/s, done.  Total 43 (delta 6), reused 0 (delta 0)  remote: Resolving deltas: 100% (6/6), done.  To https://github.com/michaelbuchanan23/sqllesson.git  \* [new branch] master -> master |

***5/23/2018 (Day 3):***

* SQL server 2017 google search will yield lots of results on the topic as will searches for C#
* Coding from scratch helps ingrain things better than copy and paste so code from scratch where possible
* Create apps in 2 steps:

1. Make it work/***FUNCTIONAL*** no matter how ugly it looks; and
2. Make it ***ELEGANT*** meaning:
   1. Robust meaning it doesn’t break;
   2. It is easily enhanced by someone else; and
   3. It’s efficient

* How do we keep learning:
  + Meetups
  + CLEs
  + Pro-bono work
* Make sure to verify your understanding of what an individual wants in their program
  + Verify by reflection – repeat back the request in your own words
* Join customer and order, sort by order amount biggest first so desc, no id fields but all other columns in
* Functions page for strings on Microsoft website:
* 
* When doing a delete make sure there is a where clause or it will delete everything…use where clause even if you intend to delete everything
* You can highlighted certain portions of code in Sql Server if you only want to run that particular portion of the code
* Comment group of lines shortcut key is ctrl+k+c
* Uncomment group of lines shortcut key is ctrl+k+u
* Read the data you want to change first via UPDATE with a where clause then copy that WHERE clause to your update in order to accurately update the data
* Don’t ever update your primary key
* The only time (usually) you should use null with numbers is when using foreign keys
* nvarchar is now the new way to define a string because of international languages where you need a double byte
* zip codes should always be strings because it doesn’t represent a mathematical number

***5/24/2018 (Day 4):***

* *Interviews*: Be succinct during interviews in your answer unless you are asked to elaborate

***SQL Server:***

* When changing the table definition the syntax is ALTER TABLE [name of table]
  + Usually used to add/drop column or change the type (e.g., change zip code from int to string)
* Purchase Request System (PRS) Tables:
  + User: list of users authorized to log in. A user is the owner of a purchase request.
  + Vendor: company that sells products
  + Product: an item that can be requested by a user on a purchase request
  + Purchase Request: grouping of all items a user requests (i.e., instance of a request containing user and total)
  + Purchase Request Line Item: a single product and quantity on a purchase request
* To create a unique inex on a table:
  + Create (unique) index [name] on [tablename](col1,col2,…)
* Stored Procedures:
  + Basically creating functions for sql
  + You may also just do queries in stored procedures (in cases like where variables are needed like different users production per week which will change with each query meaning users and the week will change with each query)
  + Stored procedures vary wildly across different database systems (i.e., MySQL vs. SQL Server)
  + Code block is the same syntactically as one line of code – between begin and end of stored procedure is a code block
* Create stored procedure to load data into tables for each of our new tables called “Add” and “Table Name”

***5/25/2018 (Day 5):***

***TO DO: COMBINE NOTES INTO ONE DOCUMENT***

***GIT/GITHUB Assignment Information:***

* We will push our assignments to greg via <https://github.com/gpdoud/dotnet4-assignments.git>
* “git pull origin master” is how we pull
* If we are having issues with pushing then we should pull down the most recent version of the folder via “git pull origin master” then attempt to “git push origin master” again

***Code blocks:***

* Is either a single statement or
* Begin…End

***T-SQL:***

* Allows one to code procedures in sql server

***General Coding Note:***

* If you are nesting your if statements more than 3 levels deep then you need to reevaluate how you’ve done it and do it more efficiently and easier to read
* When you need to know whether a certain kind of data exists then you use an Exist – 1 statement such as in “Select1-LookFor1InstanceAndReturnTwoDifferingSentences(UseForTransactions).sql”
* Looping does a group of statements potentially more than once until some Boolean condition is true or false
* Default values in SQL are all null

***5/29/2018 (Day 6):***

* Ask Greg how to go about solving a problem like Practice>SQLBasic Join>Challenges on [HackerRank](http://www.hackerrank.com/) – basically, how to solve difficult problems on hacker rank without looking up an answer and attempting to reverse engineer it?
  + Reverse engineer it then attempt to solve it again a few days later
* How to implement row\_number() in sql?
* How to implement pivot in sql
* Look into SQL diagramming techniques which will help us understand how to put things together

***C#:***

* C# is the way to program for windows
* We need to be as good in the IDE(i.e., visual studio) as we are with C# the language
* To familiarize ourselves with Visual studio make sure to right click on everything in visual studio
* SQL is declarative meaning you tell it what you want and it does it for you
  + In C#, it is a compiled language meaning you have to tell it how to do it unlike in SQL
* It is IMPORTANT that we focus on the concept of how to program and don’t focus on the syntax type of things
* Compiled languages
  + *Source:* Start off with your source (i.e., lines of text that you enter in your program)
  + *Compiler:*
    - The source goes into a compiler to compile the code into intermediate language (aka byte code—it is also close to assembly language)
    - The compiler shows syntax errors in the source code
    - Note that the intermediate language is not output if there are errors
  + *Runtime Environment:*
    - Takes the byte code/intermediary language and interprets it to run
    - Semantic errors/logic errors are errors between intermediary language and runtime environment
      * There errors are like bugs in that they may be sporadic – need to recreate the error to know what is wrong and how to fix it
* *Classes:* classes are allowed to have 2 things in them: (1) properties; and (2) methods
  + *Properties:*
    - Properties are things like fields, variables, etc. (thing of them like nouns—person,places,things)
    - These properties holds data like strings, integers, etc.
  + *Methods:*
    - Methods are functions, procedures, etc.
    - Code that does something…actions
    - Typically called the actions on the properties
* Put everything in main before it closes because after main runs it will shut down the program
* Capital letters are typically used for programs, methods, etc. in C#
* When creating classes think about the thing that you are modeling…what do I need to know about it in order to do this modeling…so step 1 = add properties to the class
* “void” means the method does not return a value
* Creating classes:
  + Properties remain private
  + Methods are public
* Data types:
  + Scalar (int, bool, DateTime,Decimal…)
    - Cannot be null by default – recently they have allowed null to help with sql issues where nulls are allowed in these data types
  + Reference (everything else) includes strings
    - Can be null
* “int[]” is an array of integers whereas “int” is just an integer
* “struct” creates a new type that is a collection of other types
  + Note that struct does not have to contain the same type while arrays must all be the same type

***5/30/2018 (Day 7):***

***Git -> Github in Visual Studio:***

* In visual studio to add a repository go to Team Explorer -> Settings -> Repository Settings -> Remotes -> Add and add as the window below shows



* Then to commit go to Changes and enter a message for your commit, do a commit all then sync then push it remotely
* Sourcetree = gitbash but with a GUI
  + Greg stated some people he knows use this instead of gitbash

***Commenting code:***

* Comment your code so that someone else can pick up and figure out what you are doing
  + Can do so by “//”, “/\*…\*/”, and ///
  + The three slashes can turn your documentation into a webpage and according to Greg is “the way to go” use <summary> …. </summary> for purposes of tagging the summary of your page for a website
  + Also use <param name = “args”>Command line parameters as strings</param> -- didn’t catch what this was for
* To comment out lines you can highlight them then click then “ctrl+e,c”
* To uncomment lines you can highlight the block of code then “ctrl+e,u”

***C#:***

* Console.ReadLine() returns a string
* If statements:

If (Boolean expression)

Statement | statement block

Else

Statement | Statement Block

* Always use curly braces for if statements/statement block even if it is only one line of statement
* Ctrl+shift+b in visual studio does a build
* Don’t nest more than 3 if statements --- if you need more than that then use
* While statements:

While (Boolean expression)

Statement | statement block

* While statements keep looping through until the Boolean expression returns false
* Using “break;” in a while statement will break the while statement even if its Boolean expression is still true

***C# Methods:***

* Blocks of code that execute anywhere in your program to perform a function such as changing string to integer
* “Main” is a method in “static void Main(string[] args)”
  + The method signature includes the name of the method, whether it returns anything, and the parameters
* It is generally bad practice to put one method inside of another method
* Put created methods above or below the “Main” method
* Methods may share the same name (called overloading) so you can pass differing parameter types to the method and get the result you want (e.g., a method called “Area” that can calculate areas of squares, rectangles, and circles depending on the type of input it receives)
* ***Scope:***
  + When you define something it is only available in that scope (e.g., variables declared in the main method are not available in other methods)
  + If we create variables in things like nested if statements then they will not exist outside of the statements they are in so it is better to create those variables before running the if statements
  + Wherever your variables are declared that is the scope so whatever that code block is
* Random #’s:
  + Random rnd = new Random();
  + > int nbr = rnd.Next(1,7) – give me a random number between 1 and 6
    - [Random.Next Method (Int32, Int32)](https://msdn.microsoft.com/en-us/library/2dx6wyd4(v=vs.110).aspx)
* Method – breaking up
  + You should break up methods
  + A method should only do one thing
  + You should be able to see every line of code of method on your screen (roughly 20 lines)…break up into smaller pieces if they get any bigger than this
  + The best programming right now is being “loosely coupled” (kind of like modular)
* To find bugs in code you can “step through” the code using visual studio:
  + Debug -> Start debugging (F5)
  + Debug -> Step Into (F11)
    - steps into a method call
  + Debug -> Step Over (F10)
    - steps over the method (i.e., execute the method and go to the next statement)
  + highlight a line of code and right click and run to cursor will run the program to that line and then stop

***5/31/2018 (Day 8):***

***Class:***

* A ***CLASS*** is a user defined type such as int, string, bool, etc.
  + So if we create a class named “Student” then “Student s” would create a variable “s” of class type “Student”
* The class name has to be unique in the namespace
* There are only two types of things that can be in a class:
  + Properties (data/fields/variables); and
  + Methods
* The Class functionality was created because:
  + When something breaks you only have to go one place to fix it versus going everywhere that the code has been copy-pasted --- in other words it isolated pieces of code
  + it also helps simplify complex systems which would then help with debugging
* Put all your class files in its own source file…the name of the source file should match the class
  + To create a new source file:
    - Go to solutions explorer, right click it, go to add, and then class
    - Shift+Alt+C also works
    - Either way you do it above, after the window opens to add a class for this class we want “Class --- Visual C# Items”
* Process to create classes:
  + What data do we need?
    - For example, if a square then:
      * We need the area (length of side2) or perimeter (4x length of side) which can be found by getting the length of one side
* [Classname] [variable] = new [classname] ();
* Interpolation ($”{a}”) – allows you to use variables in the quotes by putting dollar sign at beginning of Console.WriteLine
* ***ENCAPSULATION***– bringing together the data and the methods that operate on that data into an object called a class --- Greg said to know this for interviews
* Use verbs when creating methods telling others what they do
* Coding challenge:
  + Let’s Make A Deal 1,2,3 --- behind either curtain 1, 2, or 3 is a grand prize
  + You pick a number 1, 2, or 3
  + A number that you didn’t pick is revealed as not being the grand prize
  + You are now given the option to change your guess…should you change it?
  + Now write a program to simulate it
* For debugging Greg uses “quick watch” in visual studio a lot to check the state of variables – you can get there by putting your cursor after ending “}” for “static void Main(string[] args)” then doing “Run to Cursor” then going back and right clicking on a given variable and going to “quick watch” via right click
  + Great for checking error conditions to make sure it works when it’s supposed to
  + Get familiar with the debugging tools in visual studio so we can quickly fix problems with our code
* ***Arrays:***
  + Use List and Dictionary for arrays – usually list – list is the same as a fixed array except it is dynamic
  + In a dictionary you assign each item in the array a key and a key value (e.g., key = OH and value = Ohio)
  + List <Student> says we are going to have a dynamic array of student instances—is an index of data
    - public List<Student> Students = new List<Student>();
  + to add things to a list use:
    - dotnetbc4.Students.Add(foster);
  + to remove things from a list use:
    - dotnetbc4.Students.Remove(foster);
* ***Breakpoint:***
  + Click on the grey line at a line of code
  + When debugging it will stop the program running on that particular point

***6/1/2018 (Day 9):***

***Classes:***

* List<int> -- tell it what data type will be in the list, in this case it’s an int but can also be a user created class like Student
* Dictionary <string, string> --dictionary has two parameters
  + Can’t change it once you define it
* Note that there is a generic class for stack – he says he rarely uses this in his career
  + It is a last in, first out (LIFO)
* Scalar variables/types cannot be null in C# as they can be in SQL
  + However, recent changes have allowed us to make scalar variables nullable but you have to declare it as such:
    - “int? k = null;” //this allows the variable k to be null and thus the following is ok
* For lists:
  + List <T>
    - Add: Add(value)
    - Update: [0] = value
    - Delete: Remove(value)
* If you are unsure if a method will return something just make it “void”…you can always go change it later
* Key mapping (keymapping) in Visual Studio: Tools -> Options -> Environment -> Keyboard

***Exceptions:***

* In the exception class Greg said to pay attention to the “Message” because that will give you a hint as to what is wrong
* When the message doesn’t tell you what you need sometimes the “Inner Exception” in quick watch will tell you what you need to know
* You get there by clicking on “View Details” after doing our “Run to Cursor” at the last “}” in the program.cs
* Try {

[statement]

} catch (Exception ex) {

[statement]

}

* For example/throwing an exception:

try {

int x = 0;

int y = 1 / x;

} catch(Exception ex) { //can also use the type of DivideByZeroException

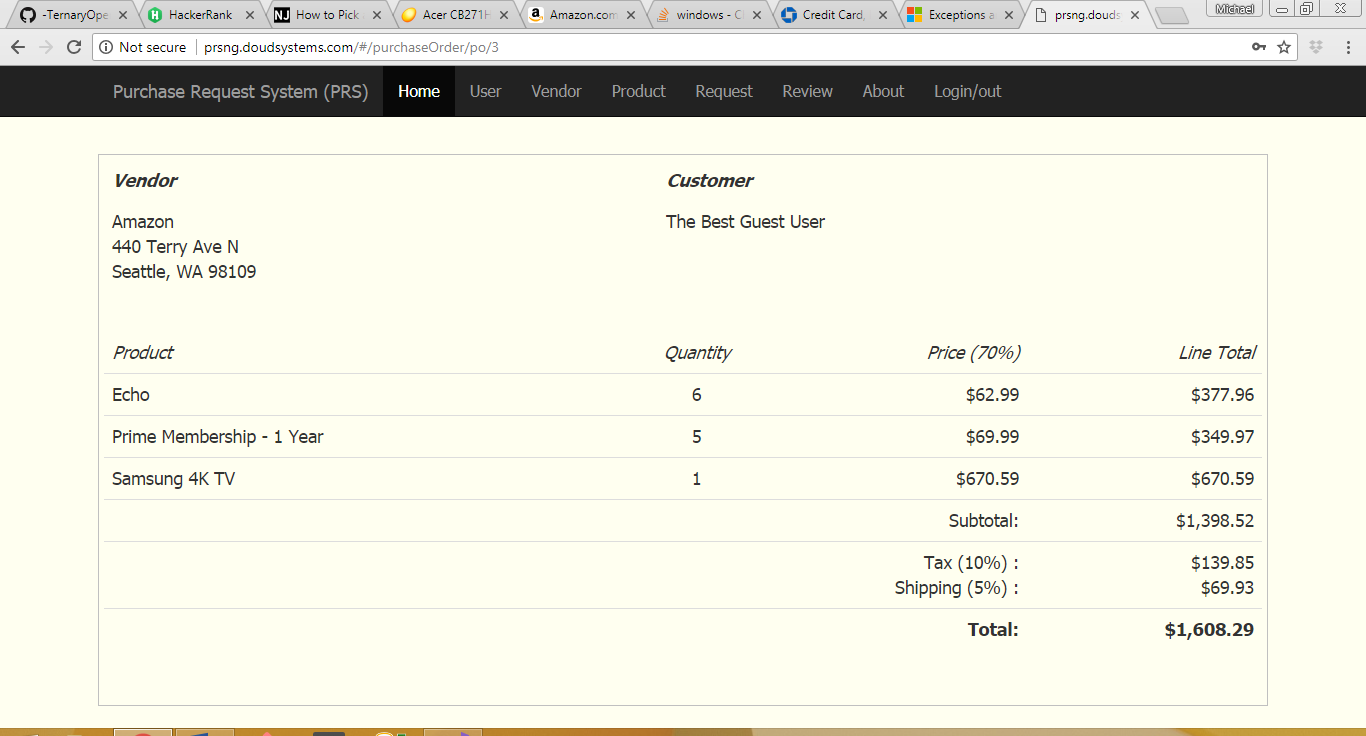
Console.WriteLine("Can't divide by zero bro");

}

//outputs “Can’t divide by zero bro”

* There is a generic application exception you can put in your own program called “Application Exception”
* When to use exceptions: often in else statements after everything else has been looked at…in other words, use exceptions for error checking in an “else” statement with an “if” statement where other error checking is happening – useful because we can’t always think of everything that could go wrong and good exception handling can help you track down where the errors are happening

OPTION PIECE OF WORK FOR CAPSTONE THAT NO ONE HAS EVER DONE: IN THE VENDOR LIST THERE IS A SHOPPING CART WHICH CREATES A PURCHASE ORDER THAT WOULD BE SENT TO AMAZON FOR ALL APPROVED PURCHASE REQUESTS – MOST OF THIS WORK HAS TO BE DONE IN C# -- IT IS LIKE THE PICTURE BELOW -- [LINK](http://prsng.doudsystems.com/#/purchaseOrder/po/3) – UN: user PW: user



***6/4/2018 (Day 10):***

* If a product has multiple vendors then you need a product table, a vendor table, and a relational/relationship table in order to house all the relationships for product 🡪 vendor and vendor 🡪 product – these relationship tables would contain a foreign key for product, foreign key for vendor, and possibly quantity and price
* Use a password manager for passwords to avoid getting hacked

***Switch statements:***

* Pg. 16 of *A Detailed Approach to Practical Coding* by Nathan Clark
* Cleaner way to write a load of if-else statements
* Switch(v){

Case “a”: //same as saying if variable(v) is equal to “a”

[statement]

break;

Case “b”:

[statement]

break;

default: //this is essentially the “else” clause of switch-case

[statement]

break;

}

* If you need to look for >, < then you will need to use if, else statements…the switch-cases are geared toward when you have a single piece of data and you want to compare to
* Switch is also fast and easier to comprehend/read then a large if-else

***Number Functions:***

* Pg.48 of *A Detailed Approach to Practical Coding* by Nathan Clark - math functions reference

***String functions:***

* Pg. 62 of *A Detailed Approach to Practical Coding* by Nathan Clark – good reference

***Enum:***

* We can create these data types ourselves
* There are not many circumstances where we use this anymore
* We will likely run into this in legacy software/code
* One example would be something like “enum TransType” where 1 = send and 2 = receive
* enum TransType {

Send, //system assigns a value of 1 to Send

Receive //system assigns a value of 2 to Receive

}; //this statement defines Send and Receive

* the difference it makes is that we can now use an if statement such as “if (TransId == TransType.Send)” in our if statement instead of “if (TransId ==1)” –this documents our code a heck of a lot better – also this assumes that we are receiving TransId with a value of 1 or 2 where 1==send or 2==receive

***Stoplight program:***

* Stoplight: we build a solution called “Stoplight” -- this used something called workflow which Greg said is used often in programming
* This used enum types and switch - case

***Static:***

* Static means relating to the type itself, rather than an instance of the type
* You access a static member using the type name instead of a reference or a value, e.g., Guid.NewGuid().
* In addition to methods and variables, you can also declare a class to be static (since C# 2.0).
* A static class cannot be instantiated and can only contain static members
* Constants are a good use of static data (e.g., Pi)
* Static variables are like global data (meaning there is only one of them
* Use a static variable to generate an Id

***Constructors:***

* Constructors are methods that are defined in a class and get called automatically when you create a new instance --- they have a special syntax
* In our case we used them to implement a counter and set an Id for each instance
  + See StaticExampleSolution for example that we did in class
* Can make more than one constructor that take different parameters as long as the parameters that they take are unique

***DECIMAL*** is a fixed type, precise data point (ideal for currency)

Vs

***DOUBLE*** is not ideal for currency because you have to round to the decimal place you want whereas decimal will do that for you

***Inheritance (Derived Class: Base Class):***

* The example Greg gave is how a German Shepard different from a dog – he stated a German Shepard has a long snout
  + Class GermanShepard : Dog {

[statement block]

}

* “Class Square : Rectangle, Quadrilateral”
  + This is simply saying that Square inherits the properties of both Rectangle and Quadrilateral
* When one class inherits from another the inheriting class receives virtually anything that is not private from the inherited class – if it’s private then the inheriting class doesn’t get it

***If you click on something like “integer” and hit F1 then it will take you right to the Microsoft page for that data type***

***Types of Classes:***

* ***PUBLIC*** - it is accessible outside of the class that it is defined in
* ***PRIVATE*** – accessible inside the class only
* ***PROTECTED*** – accessible inside the class and all inherited classes (i.e., same as private but public to inherited classes

***Polymorphism:***

* This is where the virtual—override stuff comes in to choose the right print method, for instance, based on what the type really is vs. what variable we place it into – see the “InheritanceSolution” for anexample of this
* Calls different methods based on what it is at runtime
* The 3 tenets of object oriented programming (OOP):
  + ***ENCAPSULATION*** - bringing together the data and the methods that operate on that data into an object called a class --- Greg said to know this for interviews
  + ***INHERITANCE*** - When one class inherits from another the inheriting class receives virtually anything that is not private from the inherited class – if it’s private then the inheriting class doesn’t get it
  + ***POLYMORPHISM*** - Calls different methods based on what it is at runtime

***Banking:***

* Account
  + Savings : Account
  + Checking : Account
* Account has:
  + nbr (number) //string –account# won’t change so static?
  + owner //don’t worry about adding the owner at this time
  + balance //decimal data type -- private
  + description //string
* Account needs methods for:
  + Deposit //return decimal of updated balance
  + Withdrawal //return decimal of updated balance
  + GetBalance //return balance so decimal return type
* All Savings: Account have:
  + IntRate (double type since more than 2 decimals may be needed)j
  + Method for CalcInterest() //do simple interest
* All Checking : Account have:

***6/5/2018 (Day 11):***

* When you create a set of classes for a customer to use don’t use Console.WriteLine, rather use exceptions because they may be running on different platforms
* Always create a default constructor first when creating a new class – default meaning a constructor with no parameters
* When you have a method in a base class that will need to be used in a derived class differently, then mark it as “virtual” and mark the method in the derived class “override”

***6/6/2018 (Day 12):***

* A class is composed of 2 things: properties; and methods.
* A class can only inherit from 1 class in both C# and Java

***Interface:***

* An interface is often called a contract
  + ***CONTRACT*** -- 2 things come to an agreement on how they will work together
* We create interfaces same way as classes so IPrint below
* Example in class was for a print method – Print(string, bool, int, DateTime)
  + Create an interface that would define the print method above
  + Interface would look like below:

Interface IPrint {

Void Print (string, bool, int, DateTime);

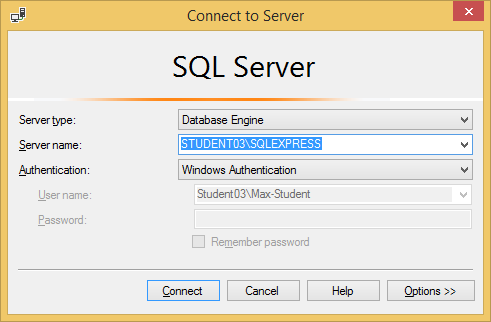
}

* To create an interface go to “Add New Item” and select an “Interface”
* An interface, like a class, is a type
* You can’t execute an interface directly, you have to execute it with a class
* When we create our backend we will be using IEnumerable and we will need to make the call on using either a static or dynamic array – IEnumerable is in InterfacesSolution
* We will not be writing interfaces in boot camp but we will be using one
* Remember there is no code inside of an interface…only definitions of methods
* When you instantiate an interface you can only do so in the InterfacesSolution program via the Printer class so “IPrint printer = new IPrint();” won’t work but “IPrint printer = new Printer();” will work
* To inherit interfaces and classes use the following syntax (also note that you can inherit multiple interfaces:
  + Class [Class Name] : [Class Name], [Interface Name] [Interface Name]
* Greg said most software developers don’t understand interfaces so this could be a plus for job interviews
* System.Diagnostics.Debug.WriteLine(s); //this writes to the "Output" tab below the “s” value when running in Debugging mode (F5)
* You should use an interface anywhere you can instead of inheriting a class
* Interfaces basically say a method should be done but it may be done differently in Java and C# and thus we will have a way to do said method in C# and a way to do said method in Java
* ***INTERFACES*** are contracts that define the methods but not the implementation of those methods
* Interfaces can implement other interfaces. A class might include an interface multiple times through base classes that it inherits or through interfaces that other interfaces implement. However, the class can provide an implementation of an interface only one time and only if the class declares the interface as part of the definition of the class (class ClassName : InterfaceName). If the interface is inherited because you inherited a base class that implements the interface, the base class provides the implementation of the members of the interface. However, the derived class can re-implement the interface members instead of using the inherited implementation.

***Combining C# and SQL Server:***

***SQLConnection:***

* This is a class in C# -- used via “using System.Data.SqlClient;”
* SQL needs to know the server name
* For example, in the below image the server name is: “STUDENT03\SQLEXPRESS”



* string connStr = @“server=[server name]; database=[name of database];Trusted\_connection=true”;
  + The @ sign allows us to not have to use an escape sequence in order to insert the backslash
  + Use a constructor that uses the above sequence to connect via the SqlConnection class
  + SqlConnection conn = new SqlConnection(connStr); //string that connects to the server
  + conn.Open(); //opens the connection
    - to check that the connection is open:

if(conn.State != ConnectionState.Open) {

throw new ApplicationException(“Connection did not open”);

}

* We wrote a program about this titled:
  + CSharpToSqlSolution
* Pro tip: Run your sql statement for C# in sql server first to make sure that it works
* Make sure to handle all potential errors where a value can be null in sql and not be null in c#..best practice is to add default value via an if statement in c#
* Greg makes variables lowercase that are temporary and don’t last very long (e.g., in a while statement)

***6/7/2018 (Day 13):***

* We are going to cover a few days of Java (we will still need to learn the libraries/classes on our own) because we are a few days ahead of schedule
* Yesterday’s work on connecting C# to SQL was a reference for us to refer back to if we need it -- “CSharpToSqlSolution”
* Today we are going to build a library which makes for more efficient coding (the example Greg drew on the board was 3 applications talking to a SQL database with a library in between so that functions/methods can be reused
  + Apps are .EXE files
  + Class libraries are .DLL files (dynamic link library)
  + It is also okay for libraries to use other libraries (i.e., DLL 🡪 DLL)
* We can use the “CSharpToSqlLibrarySolution” as a template for a lot of our other work
* Our solutions will always have more than one project
* Right-click on solution part of solution explorer, go to add, new project, class library
* Once you add a class delete the default “Class1.cs” that is created
* Adding a new library class to a program:
  + Add a reference to use a new project-class (Add reference in main project via the references folder then select Projects-Solution and select your newly created library)
  + Add a using statement at top of program
* When creating a library we always need to make it public so that other programs can access it
* In the controller apps (e.g., UsersController) there are generally 5 methods that you need:
  + List() [all rows] – returns all of the instances of the class (think “List<User>”)
  + Get(id) – returns 1 user instance by primary key
  + Create – Insert in SQL (take 1 user instance) –return bool value of did it work or not?
  + Change – Update in SQL (take 1 user instance) -- return bool value of did it work or not?
  + Remove – Delete in SQL (take 1 user instance) -- return bool value of did it work or not?
* **MAKE SURE TO SHOWCASE “CSharpToSqlLibrarySolution” TO POTENTIAL EMPLOYERS**

***REFLECTION*** provides objects (of type Type) that describe assemblies, modules and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them. For more information, see Attributes.

***6/8/2018 (Day 14):***

* We started the day watching a video from CS50.net which was lesson 0 it was Harvard University’s introduction

***SKILLS TO WORK ON TO ADD VALUE FOR EMPLOYERS:***

* Angular – knowing this is a leg up because not many people know
* ASP.Net -- run totally off a server; makes for better apps nowadays
* SQL Server -- must know this well; make sure you have a solid foundation
* Entity framework -- probably only need the basics here for a programmer
* JavaScript – Greg said he didn’t do much of this because you can’t count on people’s computers to run well – JavaScript isn’t any more difficult than C# -- there are some differences to C# but not really any more difficult
* MVC – pretty important but there isn’t a ton of knowledge you need to learn MVC
* C# -- know the libraries as well as the syntax
* BE STRONG ABOUT WRITING BUSINESS APPLICATIONS SUCH AS THE “CSharpToSqlLibarySolution” we worked on 6/7/2018

***6/11/2018 (Day 15):***

* We are going to start Entity Framework today
* In an interview Greg would ask questions like “What is the most complex C# program you have ever written?” || “Why was it complicated?” || Greg said he didn’t care about school projects.
* Add something to your portfolio that is fairly complex like pig dice to make it more and more robust and describe in detail how you made it better, more efficient, etc.
* Also make sure if it’s on our resume that we can at least talk some about it such as SQL

***Entity Framework:***

* Open source technology driven by Microsoft – similar technology called Hibernate for Java
* Is an ORM (Object Relational Mapping) tool
  + Commit ORM to memory because we may be asked what ORM stands for
* Entity framework sits between C# and SQL and basically makes communicating between the two easier
* Entity Framework can be used in two modes:
  + Database First
    - Existing database: creates classes from existing database so no hard coding is needed
  + Code first
    - Creates databases from C# class code
* DBContext
  + A class that we will create where we point to all of the entity framework objects that we want to use in our application
  + We can use Entity Framework to do the DBContext
  + Include tables that we want for the application that we are writing
  + DBContext is an Entity Framework object
  + class AppDBContext: DBContext
    - Our classes will inherit DBContext
    - In our code we will create a variable such as “AppDBContext db = new AppDBContext();” – everyone calls this variable “db”
  + You will use “db.\*” to call all of your SQL commands such as “db.Users.ToList();”
    - So db is the database, Users is the table, and ToList() is the method of what to call (i.e. the SQL query)
  + Use LINQ to search for queries and the like LINQ and Entity Framework work together
    - LINQ stands for “Language-Integrated Query”
* Creating the database:
  + **Step 1:** Add Entity Framework to your project
    - What we have been creating as classes such as User and Vendor we will now be calling those models for the tables where they will ultimately be used
    - We will create the model in code, and then Entity Framework will create a database table for this model—EF will generate all of the SQL statements we need to create that table
    - If we need to add things later to the table then Entity Framework can go back and recreate the table based on the changes
    - There is a table called migration history which tracks changes in the table
  + **Step 2:** “enable-migrations” -- command using power shell
    - This creates a migrations folder which tracks table changes – we can roll back changes via this tool
    - This is similar to “git init” in that you only do it once
    - Every time you change the model you issue these two commands:
      * “add-migration “[message]”” – creates another source file in the migrations folder
      * “update-database” – updates the database
  + We are going to use Entity Framework 6 in class
  + Package manager console – view 🡪 other windows 🡪 package manager console
    - We want entity framework in the class library and not the main program
    - We used package manger to add/install entity framework via the following command:
      * “install-package entityframework”
  + We then went on to create a table via entity framework in Visual Studio
    - I don’t think I have all of the steps written down unfortunately but it went something like below – this solution was named “PrsWithEfSolution”
      * Add new project
      * Add new class which has base class of “DBContext”
      * Reference new class in original project
      * Install-package entity framework
      * Enable-migrations
      * add-migration "Initialization" //this creates the initialization class which must have a unique name each time you create one
        + Up() rolls it forward and Down() rolls it backward
      * “Update-database” in package manager console – this creates the DB
      * We also had to add EF to our main program: “install-package entityframework”
  + “var” to initialize variable is okay but you must set it equal to something
  + Any time you change your model of the DB you have to go through a migration via package manager: (1) “add-migration “Limiting all string sizes”” (2) “update-database”
  + Validation errors: Greg showed us this by adding too many numbers for the phone number “DBEntityValidationException – Validation failed for one or more entities” – can use quickwatch to dive in to what is wrong; get to quickwatch via the “View Details” option in the error bubble that pops up—if you dive in deep down into the EntityValidationErrors--Validation Errors” dropdown errors it will tell you what data field is wrong
  + If we try to update and it fails then we may get an inner exception which we can expand to see which update fails
  + Entity Framework Code First Data Annotations : <https://msdn.microsoft.com/en-us/library/jj591583(v=vs.113).aspx>
    - DataAnnotations not only let you describe client and server side validation in your code first classes, but they also allow you to enhance and even correct the assumptions that code first will make about your classes based on its conventions. With DataAnnotations you can not only drive database schema generation, but you can also map your code first classes to a pre-existing database.
    - While they are very flexible, keep in mind that DataAnnotations provide only the most commonly needed configuration changes you can make on your code first classes. To configure your classes for some of the edge cases, you should look to the alternate configuration mechanism, Code First’s Fluent API
  + STEPS TO CREATE ENTITY FRAMEWORK
    - Create AppDbContext : DbContext class
      * Add a using statement to make the DBContext usable
    - Create model class
      * Contains things like user, vendor, product, etc.
    - Add your model class to AppDbContext as a property
    - Build
    - Package Manager(PM): “install-package entityframework”
    - PM: “enable-migrations”
    - PM: “add-migration “[name]””
    - PM: “update-database”
    - Additionally, don’t forget to add a reference to the class library if you add one here
  + Recreate an entity framework for PurchaseRequest and PurchaseRequestLineItem tonight
  + CODING CHALLENGE/PROGRAMMING PUZZLE:
    - Make it elegant and functional as much as possible
    - Find a subset of areas where a total of exactly 100,000 people lie, assuming the census estimates are exactly right.
    - The actual question is on slack and below:

|  |
| --- |
| Solve this puzzle  Can you solve this puzzle? While not a requirement, we give priority consideration to candidates supplying a solution.  The 2010 Census puts populations of 26 largest US metro areas at 18897109, 12828837, 9461105, 6371773, 5965343, 5946800, 5582170, 5564635, 5268860, 4552402, 4335391, 4296250, 4224851, 4192887, 3439809, 3279833, 3095313, 2812896, 2783243, 2710489, 2543482, 2356285, 2226009, 2149127, 2142508, and 2134411.  Can you find a subset of these areas where a total of exactly 100,000,000 people live, assuming the census estimates are exactly right? |

***6/12/2018 (Day 16):***

* Capstone project:
  + Sum up the total of the purchase order in your C# code
  + Recalculate if you add or delete something from the purchase order request
* Using entity framework:

1. Create AppDbContext inheriting DbContext
2. Create model class
3. Add model class to AppDbContext as a public property
4. Do a build
5. In Package Manager Console:
   1. install-package EntityFramework
   2. enable-migrations
6. After any changes to any models
   1. add-migration "unique message about change"
   2. update-database

***6/13/2018 (Day 17):***

***Testing and Test-Driven Development:***

* We had a slideshow on this
* Think about any and all ways to break the code and try to stop them from doing so
* Most companies Greg has worked for have developers testing the software
* Software is ready when the testing stops finding bugs
* Greg will send us a printout of the slideshow
* Process:
  + Add a test
  + Run all tests and see if new one fails
  + Write some code
  + Run tests
  + Refactor code
  + Repeat
* Write the test first before you write the code
* Visual Studio has Text Explorer provided automated testing for C#
* Add testing to a program by:
  + Add a Test type project to solution
  + Add reference to project being tests
  + Add using for namespace
  + Add classes with [TestClass] attribute
  + Add methods with [TestMethod]attribute to class
  + Add ‘Assert…’ statements
  + Make sure that your classes and program and such in main program are public
* Run tests from TestExplorer
  + Test > Run > All Tests
* Can have testing set to always test for errors when you build
* Check the pre-made tests for software once receiving to see what works and what doesn’t
* To add a test project:
  + Go to solution and right click
  + Add test – Unit Test Project
  + Can test all using test explorer
    - Test > Windows > Test Explorer
    - Then “Run All” in the Test Explorer
* Typically there is no company policy on automated testing but Greg said that this is good practice to do so even where not required
* Pair programming: pair off with another developer – one person is coding and the other person is creating test cases

***Java:***

* Java uses import statements which are similar to using statements in C#
* Write line is System.out.println();
* Most methods in Java start with small letters (e.g., println)
* Current version of Java has something that is like LINQ but Greg said it is way behind .Net
* Packages in Java are similar, but not identical, to namespaces in C#
* Superclass is the class you are inheriting when creating new classes
* Syntax to inherit classes: “Public class Square extends Rectangle”
* Syntax to implement interfaces and inherit classes: “Public class Square extends Rectangle implements X”
* There is no virtual and override in Java --- it works as though you’d put that on automatically
* Best place to look for Java methods/functions is oracle’s website and stack overflow

***6/14/2018 (Day 18):***

* Greg suggested we go back and reprogram some of our C# programs in Java for practice
* SQL questions are popular during interviews such as outer joins and sub-selects

***HTML & CSS:***

* Greg gave a speech which basically boiled down to “know your audience” – corporate vs public audience – (e.g., make it pretty for public, don’t worry about corporate)
* In cmd you can use “ping microsoft.com” to find IP address for Microsoft
* In cmd you can use “tracert microsoft.com” to find path through various servers to reach Microsoft.com
* Asp.net translates Microsoft code to html/css code
* To get the https for your website you have to go through an online security company that will then give you a certificate (costs about $500 to $1,000 a year) – you will install it to your server
* Greg recommended we set up our own website with our own server for experience
* Greg recommended we know/read “The Life of a Web Page” on pg.12 of our manual
* “Ctrl + /” –comments/uncomments out a line in Visual Studio Code
* Can open developer tools with “Ctrl + Shift + I” in chrome
* “Disable cache (while DevTools is open)” was checked in our settings for “Developer Tools” in Chrome so that we are constantly getting the most recent updates
* You can change thing in the developers tools for chrome in the elements window but it won’t change the file. (do this by right click then edit text)
* All elements in html default to either inline or block
  + Span types are inline
  + H1 – H6 are block
  + Blocks get own line
  + Inline do not get own line
* Style attribute is easiest way to style our pages
  + <p style = “[Style-Attribute : Value]”>
* Serif font – times new roman, have distinctive looking letters with wings like this: T
* Sans Serif – arial or calibri which don’t have the wings like this: T
* p {color: red; border-style; solid; font-size: 12pt; }
  + Gives all p tags the color, border, and font size in the curly braces
* .mybold {color: red; border-style; solid; font-size: 12pt; }
  + Creates a class which can later be used in tags with class=”mybold”
  + Can add more classes with spaces such as class = “mybold mybold2 …”
* <P id = “para1”>
  + Can later define the style with id para1 by using:
    - #para1 { color: “red”; }

***6/15/2018 (Day 19):***

* Watched video titled “Five Things Junior Developers Need to Know S:01 E:19”
  + Learn how to learn
    - Get faster reading the documentation
    - Watch tutorial videos
  + Try to figure out problems/issues first when assigned work from you manager then run it by them and ask for support
  + Functional programming – make it testable
    - Bad:

int incr = 1;

int add (int x){

return incr + x; }

* + - Good:

int add(int incr, int x) {

return incr + x; }

***CSS:***

* Three types of styles we can create are tag, class, and id
  + Use name of tag to create style for tag (i.e., p { [styles] })
  + For id -- #p1 { [styles] }
  + For class -- .p1 { [styles]}
* To add multiple classes add a space between class names
* Selector is term for items such as #p1 and .p1 above
* Link your CSS styles via an external sheet so you can use consistent styles across a website
* Create a master stylesheet (i.e., master.css) for each website we build
* We don’t want styling to clutter our HTML so use an external style sheet and put it in a subfolder
* Greg said to define the magnitude meaning you can’t just say 2 you must say something like 2px or 2%
* We will use some “pt” for font sizes – pt=points which is a typesetting unit – this stuff is on pg.46 of our workbook
* Use relative for websites so that things display properly on different sized screens
* Create a style for the body tag which has both the font-type and the font-size
* div p { [styles statement} ]– applies only to p tags in a div & includes p’s children
  + div>p – applies only to p tags in a div and does not include p’s children
  + div+p – select all p elements that are immediately after element 1
  + for more of this grouping selectors see pg.49 of our text
* attribute selectors (i.e., [src]) can put certain styles on those selectors
* pseudo-class selectors for anchor tags are covered on pg.50
* need 3 things for “border”
  + border: [pixel size] [border] [color]
    - border: 1px solid green
* can use the element.style portion (lower half of the window on the left in Chrome) to see what elements that we add will do the page in real time
* note that we can define top, right, bottom, and left margin/padding with numbers in sequential order respectively for the margin/padding selector
  + can also do top/bottom and right/left with just 2 numbers respectively
  + if there is only one number then it is all sides
* we should put different sections of our websites into divs
  + e.g., put your contact info on a public site into a div so you can hide it as necessary – you could make someone log in in order to see it
* Greg used classes for his resume website on things like school, degree, major, gradyear and the like so they could all be styled similarly
* Use web safe fonts for your websites
  + You can use fallback fonts if your original choice doesn’t work
* Be consistent with your styles
* There are tools called “color pickers” that you can use to help find colors on logos and such
* Gradients can be used which are transitions between two colors

***6/18/2018 (Day 20):***

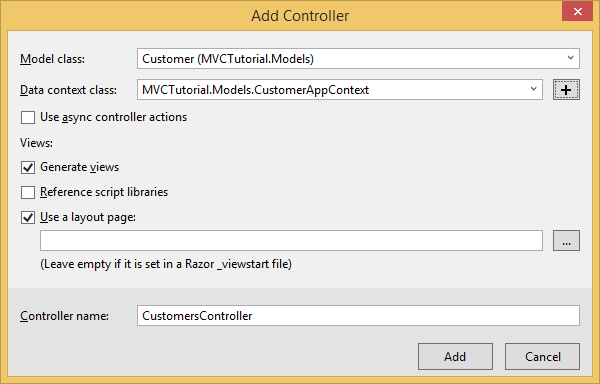
* We continued our discussion on HTML/CSS starting with module 6 (anchors and hyperlinks) on pg.78 of our manual
* Can jump to id’s using <a href=”#[id]”>
* Greg said to use the “title” attribute in a href tags so it clues the user in to what the link is for
* Note the html 5 div-like tags such as <header>, <footer>, etc. on pg.94 of our manual
* Note that there are usually problems with Internet Explorer rendering vs. Chrome/Firefox so make sure to text your stuff on IE if you need to develop for it
* Take note of the shiv file for html5shiv.js which helps render things correctly in IE versions prior to 9 (pg.95 top)
* Lossy (such as jpg) means some data is lost when you scale it down -- Lossless (such as gif and png) nothing is lost when you scale it down
  + Thus, png/gif > jpg
* Resize images as copies of the original
* Best practices for images:
  + Images should be pre-sized and not rescaled by the height and width attributes
  + Height and width attributes should be provided to reserve space for the image
  + Alt attribute contain text to describe the image…this used for non-graphical interfaces, screen reader applications, and images with broken links
* Base64 is the textual representation of an image –see pg.101
* Can use glyphys/glyphicons such as arrows and the like – Greg said there are millions of them out there you can download for free

***HTML Forms:***

* Forms are the most useful part of HTML
* Form controls are an essential part of that
* The input type can take on lots of different ways of inputting data (e.g., type=text is a text box)
* Form controls are the things the user will interact with and they are in the <form> tags
* Two methods in forms are GET and POST
  + These determine how data is passed from one page to another
  + Get means take data in the form tag and pass it on the url line
  + Post method is where url doesn’t have any data in it but rather it is passed in the body
  + Post is encrypted
  + Get can be bookmarked whereas Post cannot
  + More on pg.110 of our book
* Form controls will often have the same value for both name and ID attributes
* Input types on pg.111 of the manual (e.g., text, checkboxes, files, images, password, etc.
* Use selected attribute to default select in form
* HTML5 form enhancements listed on pg.117 such as color, date (datepicker), email (format checks), and number (which pops up number keypad on phone), autocomplete (can be toggled on/off)
* Web forms are deprecated
* We will use the Microsoft recommended way to create forms on the server

***MVC (Model Views Controller):***

* A model is a class (e.g., user class, vendor class)
* Controller: the things that you can do with your model (e.g., list, get, update, set, create)
* The view is what the user sees (i.e., rendering of the model)
  + If it renders a model then it is probably a view
* Bootstrap comes pre-loaded when we create new ASP projects
* JQuery is a JavaScript utility library which is also already built in to each of these new projects
* All of the stuff with @ signs in the various files (e.g., \_Layout.cshtml) means thing are run on the server via Razor
* Action links:
  + Link text is what is displayed
  + Action name is the method in the controller –usually index
  + Controllername is the controller it is using
* The @ stuff is razer syntax
* @RenderBody() is where the views are injected into the page
* Your views should be bound by a div statement
* IIS does the same thing as Apache and Tomcaat – windows system has a full version of IIS
* If you are just changing the HTML then you can refresh, if you are changing the Razor code then you must rebuild and rerun (Ctrl+F5)
* To add a class for entity framework on these web apps:
  + Add new class under models
  + install-package EntityFramework
  + enable-migrations??
  + Do a build
  + Add a controller under controllers as shown below



* + This should add a controller and views for you
  + Change connection string in web.config (roughly line 12)
  + Add a new LI item to the ­\_Layout.cshtml page

Greg’s cheat sheet to do this:

1. Create project (web .net .Net Framework) 🡪 ASP.NET Web Application (.NET Framework)
2. Create model (e.g., Customer [class])
3. Do a build
4. Install-package EntityFramework
5. Add a controller (right-click on “controllers” then “add” then “controller” in solution explorer) –we want MVC 5 w/EF & Views
   1. Select model which should match the one you created above in step 2
   2. Use + sign to Create data context class and change the highlighted portion to whatever name you want
6. Change connection string in web.config to match our connection string for sql

***6/19/2018 (Day 21):***

* Use module 3 (controllers and actions) in the MVC portion of our textbook as a reference

***Routing:***

* Routing means how does the controller know what is needed – it occurs at the URL
* App\_Start/RouteConfig.cs – this file tells the web pages where to route to; i.e., this decodes the URL – see routes.MapRoute()
  + routes.MapRoute(

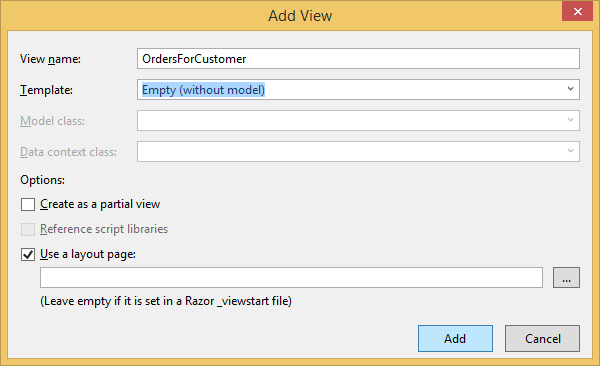
name: "Default",

url: "{controller}/{action}/{id}",

defaults:new {controller = "Home", action = "Index",id = UrlParameter.Optional}

);

* + Covered starting about pg.21 in our book
  + Greg said we may want to alter these default routes such as when a username and password is required – we would need a new method for login which we would route to when people need to login
    - /Users/Login?Username=user&password=x37
      * Greg said the above string is wordy and prone to mistakes so use the way below this
    - Users/login/user/x37
      * We can do it this way by changing the routing information
    - Note that both of the above two ways are not secure and thus our final project will not show up in the browser
* Routing is figuring out from the url what controller and method to use in order to do what we want
* View-Model
  + Just like any other model except you don’t do any updates to it
  + Is used only to view data
  + Normally has more than one table’s worth of data in it
  + Not just foreign key stuff
  + In our example we showed all orders from a chosen customer
  + We created a new folder for this called ViewModels and put a class in that folder called OrdersForCustomers.cs
  + We added Customer Customer as a property
  + We added Order Order as a property
  + Add a new ActionResult method in the Customer controller
  + Update the index view for the customer with an option for “Orders”
  + We created a new view like below by right-clicking our method name in our Customer controller and creating a view:



* No semicolons at the end of Razor syntax
* OrderLine
  + Id int PrimaryKey
  + OrderId int ForeignKey
  + Product string
  + Quantity int
  + Price decimal
  + (linetotal) //not in DB just in form and it multiplies Price\*Quantity
* The view for OrderLine
  + Order.Description
  + Order.Description
  + Shows Product | Qty | Price | Subtotal

***6/20/2018 (Day 22):***

* We discussed pg.136 regarding rules for embedding code in our manual (module 6)
  + Notably if @Model.Cost equals 3 then:
    - @Model.Cost \* 3 will display 3 \* 3
    - @(Model.Cost \* 3) will display 9
* Pg. 141 shows how to designate names to display in razor via the models class (i.e., Firstname becomes First Name)
* Pg.145 editor templates
* LINQ syntax for pulling the enrolled classes for a student; it joins 3 tables together:

cfs.Classes = from s in db.Students

join e in db.Enrolleds

on s.Id equals e.StudentId

join c in db.Clasees

on e.ClassId == c.Id

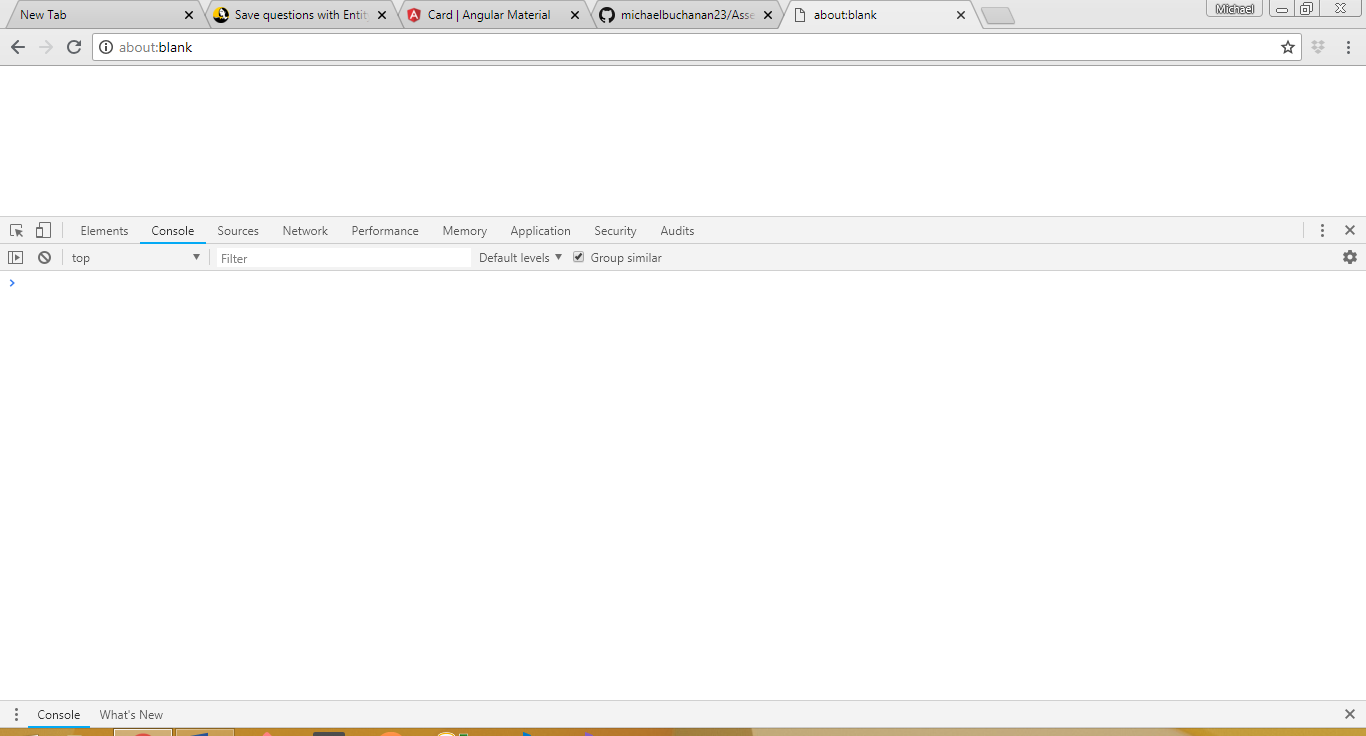
select c;

***6/21/2018 (Day 23):***

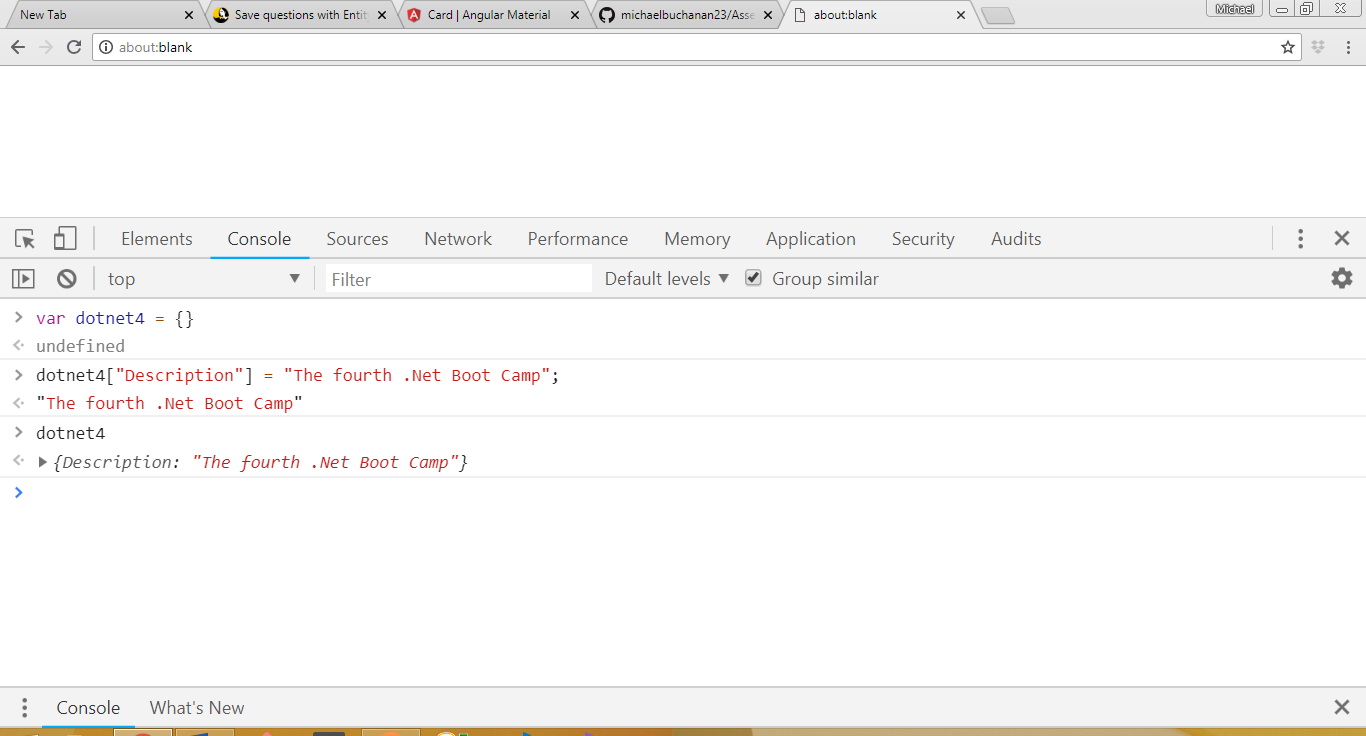
* Started day with Assessment1 which was building an EF Framework with Views/Controllers about an Owner/Vehicle
* Today we start on WEB API and JSON

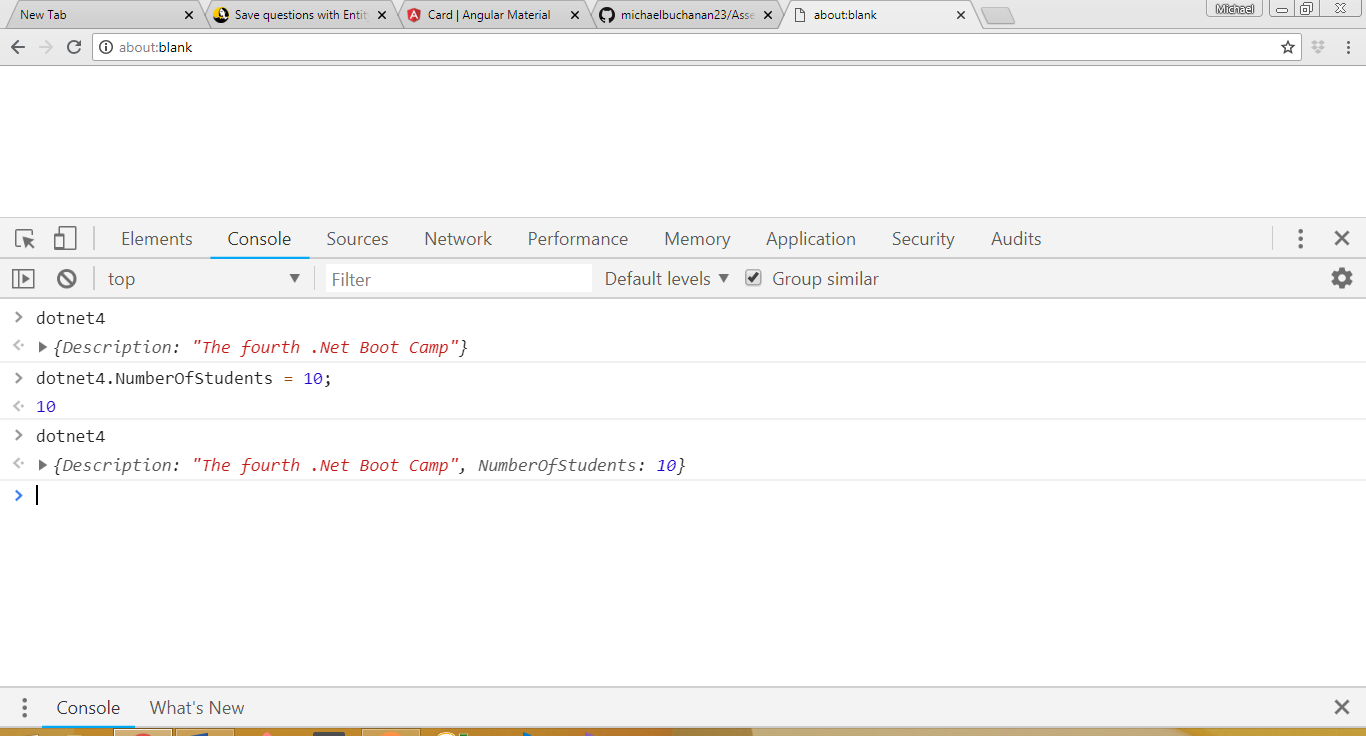
***JSON:***

* JavaScript Object Notation
* This has almost completely replaced XML
* We used developers tools(F12) console on an about:blank webpage in chrome for our lesson

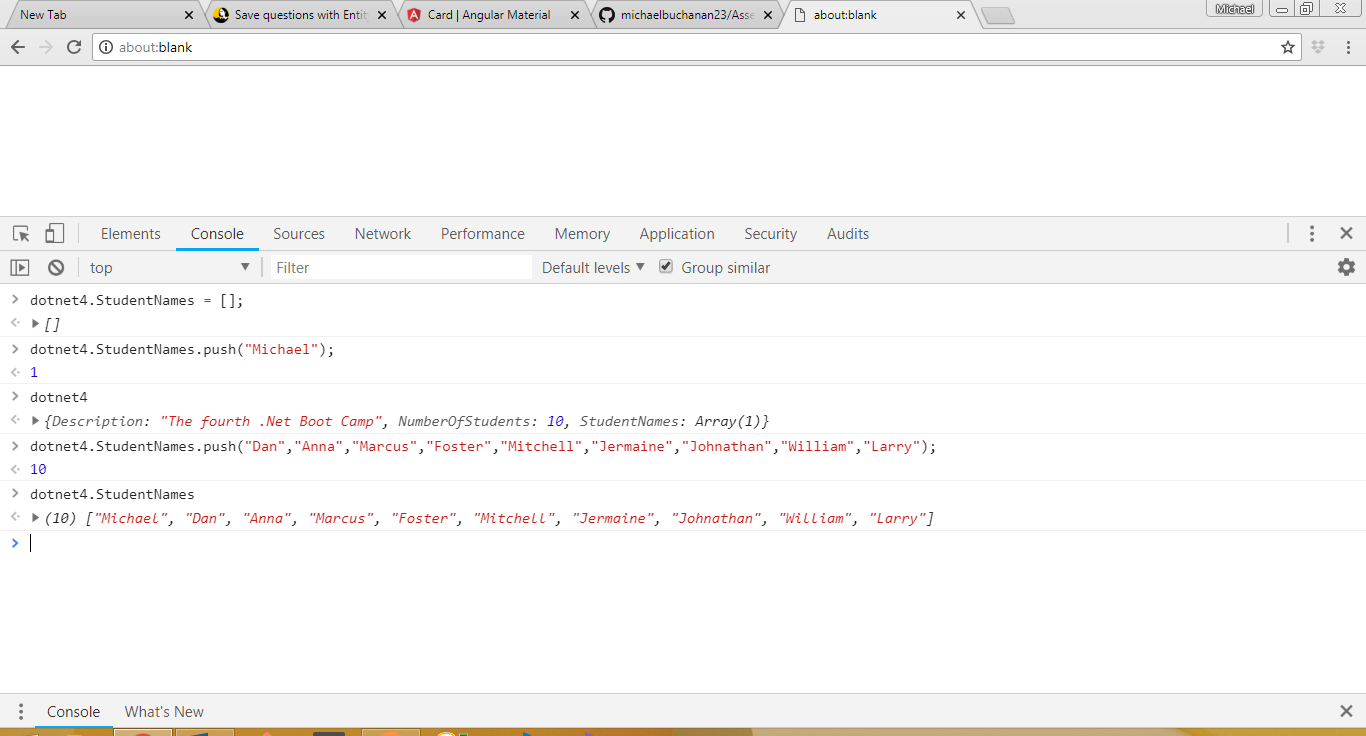


* XML replaced CSV because it was easier to read
* XML got too big because the amount of data being transferred was overwhelming and slowed things down—too big because it used opening and closing tags much like HTML does
* JSON was introduced which cut out the XML tags which greatly reduced the file size – it basically cut down on the required syntax to define things and thus less needed to be sent between servers and users to get the same information – this is all based on a key value pair
* JSON Types:
  + String
  + Numbers
  + Bool
  + Array
  + Object (which is simply JSON)





* The base array in Javascript can have things added and removed from it like a List in C#
* We call things in an array via array[index#]
* To add something to the end of the array we “push”
  + dotnet4.StudentNames.push(“Michael”)



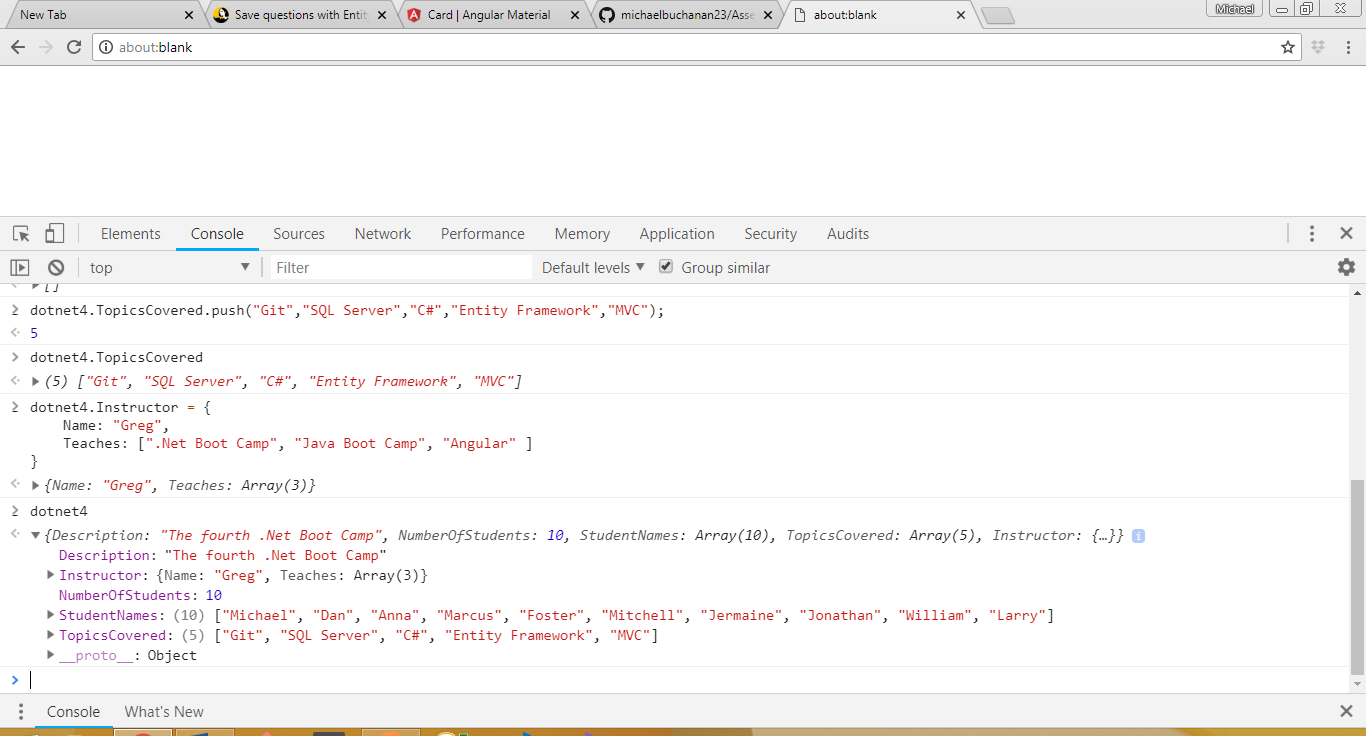
* A JSON object can itself be an object (e.g., for instance so we can have lots of details about our instructor)

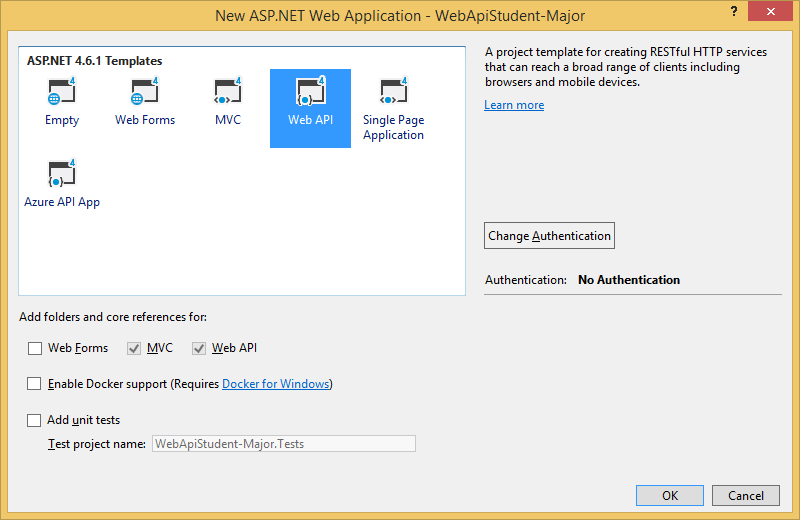
Dotnet4.Instructor {

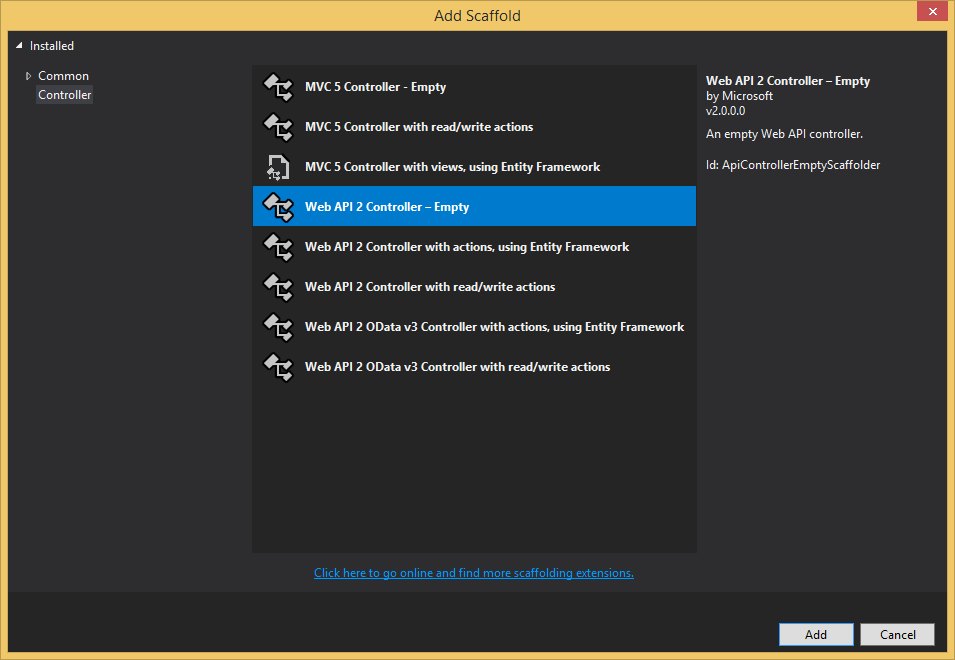
Name: “Greg”,

Teaches: [ “.Net Boot Camp”, “Java Boot Camp”, “Angular” ];

}



* Booleans are just added using true/false notation
  + dotnet4.IsAGoodClass = true;
* in JavaScript there is a foreach loop functionality that will iterate through arrays for us to display things on our pages
* 
* created a DB context in MVCWebAPIDBContext.cs in models folder
* we created a new MVCWebAPITutorial



* Customers model then webapi2 controller as above
* We had to add our own DB Context
* Reading data is a GET whereas updating data is a POST
* We made changes in WebApiConfig.cs file in the App\_Start folder
  + We added the RegularAPI by copying and pasting the API that was already there and made changes as follows

config.Routes.MapHttpRoute(

name: "RegularApi",

routeTemplate: "{controller}/{action}/{id}",

defaults: new { id = RouteParameter.Optional }

);

* Our entire block of code for WebApiConfig.cs was as follows

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http.Headers;

using System.Web.Http;

namespace MVCWebAPITutorial {

public static class WebApiConfig {

public static void Register(HttpConfiguration config) {

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "RegularApi",

routeTemplate: "{controller}/{action}/{id}",

defaults: new { id = RouteParameter.Optional }

);

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

config.Formatters.JsonFormatter.SupportedMediaTypes

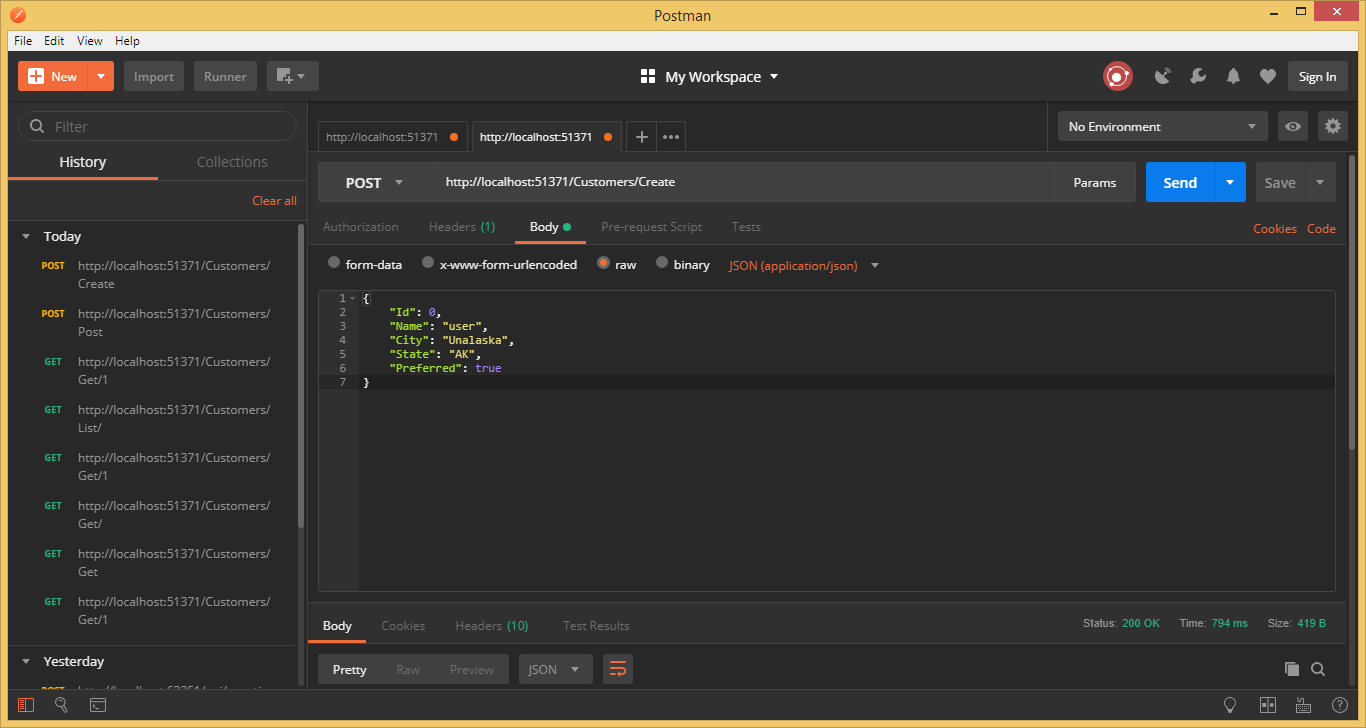
.Add(new MediaTypeHeaderValue("text/html"));

}

}

}

* The config.Formatters part at the bottom configured the data to JSON
* Used Postman to test our controllers GET/POST features – we will test our backend without having to write programs ourselves
  + If it doesn’t work in Postman then more than likely our backend doesn’t work
* Testing via adding JSON data in Postman

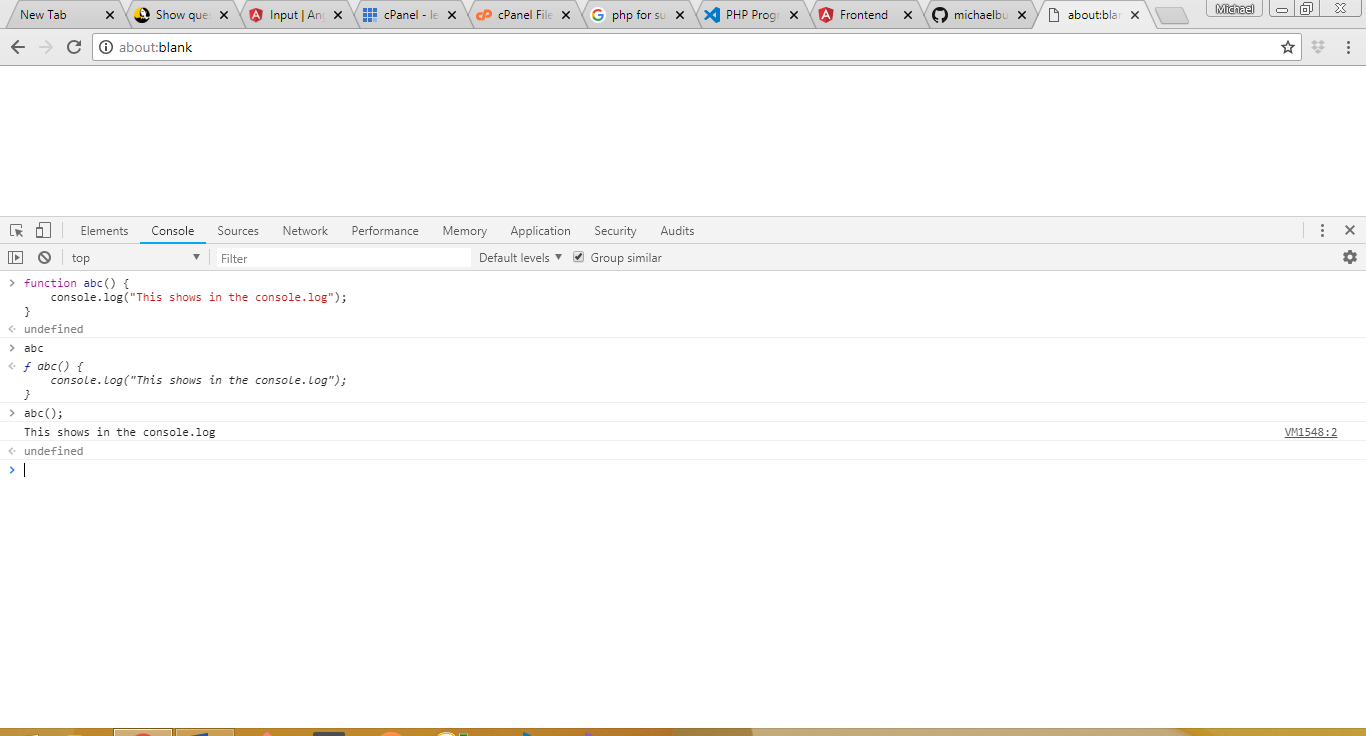


***6/22/2018 (Day 24):***

* Json.ignore will ignore certain recursive relationships such as foreign key loops
  + E.g., Purchase request with a List of purchase request line items and purchase request line items with a key to purchase requests
* We created WebAPIStudentSolution which was very similar to our project yesterday titled MVCWebAPITutorialSolution regarding Customers and Orders
* We created a JSON response for errors in WebAPIStudentSolution via a JsonResponse.cs class which we then used in our MajorsController to return data that helped understand errors (such as a string that is too long)

***JavaScript:***

* Programming language for the browser
* Similar to C#
* Greg said lately there has been a push to remove JavaScript from websites and put those loads on the Server
* JavaScript is done within HTML script tags
  + We will put our script statements in the head section
  + <script lang=”javascript”>
* Module 1 talks about editors, using Developer Mode (F12)
* C# is static (strongly) typed meaning we set the type of the variable at initialization whereas…JavaScript is dynamically typed meaning we can make var i = 0; and then change to i = true with no issues; note you can’t change like this in C#
* JavaScript allows both single and double quotes to bound
  + Greg said to bound JavaScript strings in double quotes as a good practice
* 6 types in javascript:
  1. Number
  2. String
  3. Boolean
  4. Array
  5. Object
  6. Undefined (variable that has not been assigned a value)
* We used the F12 Chrome Console as a test JavaScript playground
* Semicolons separator statements on the same line – if things are on a separate line then no statement is needed
* If we do:
  + var i = 0;
  + typeof I then we will get “number” back
* JavaScript variables are case sensitive
* Commenting code is same as C# except no “///” so commenting is // and /\*---\*/
* JavaScript has if, while, for loops, etc. much like C#
* Note that a string of “1” and an int of 1 in JavaScript will show as equal even if they are not
  + This is because JavaScript makes assumptions about things when comparing data with different types
  + For example all numbers can be turned into strings so if you add the string 1 and the number 2 it will turn 2 into a string and concatenate
* 3 equal signs (===) means the value and type must be the same (i.e. string “3” does not equal number 3) – Greg recommended we always use triple equals because it is more precise
* JavaScript functions are similar and almost identical to C# methods – major difference is in Javascript these are standalone whereas in C# they exist inside of a class

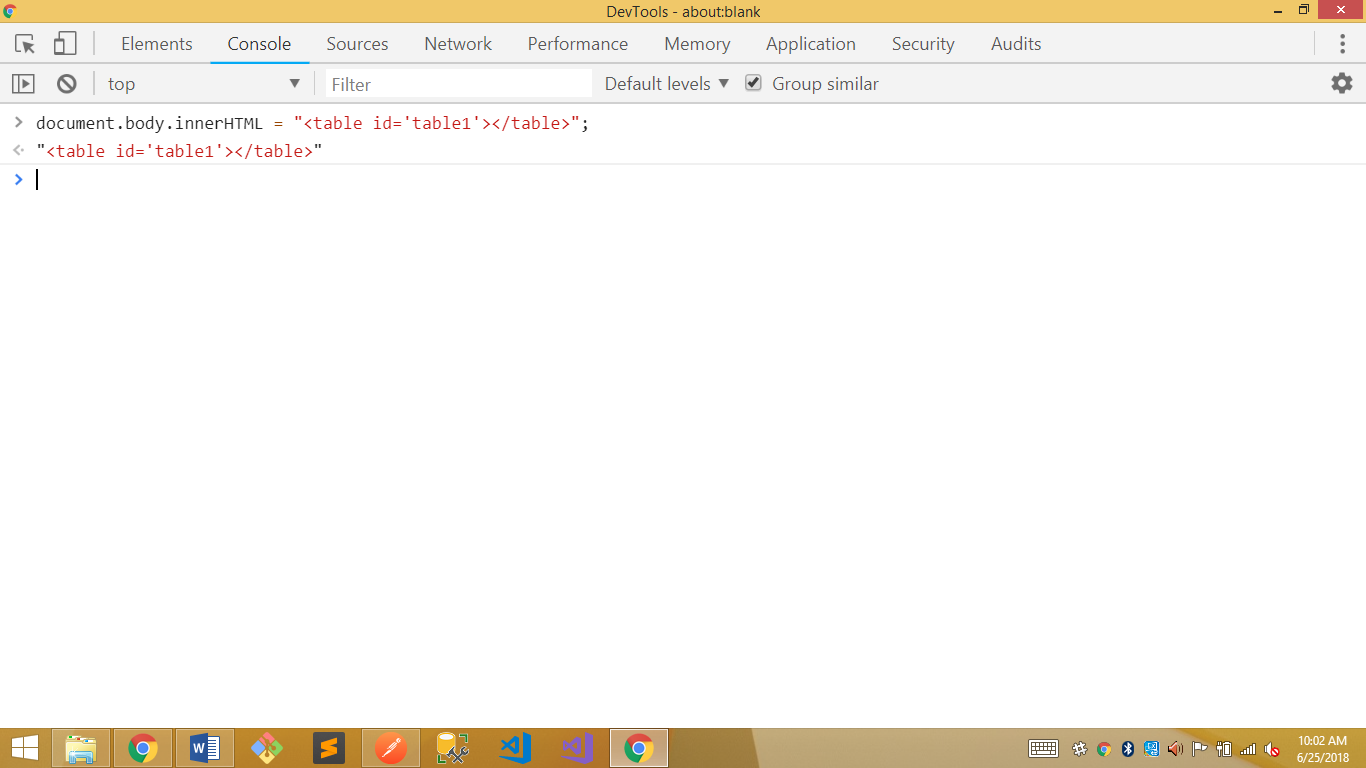


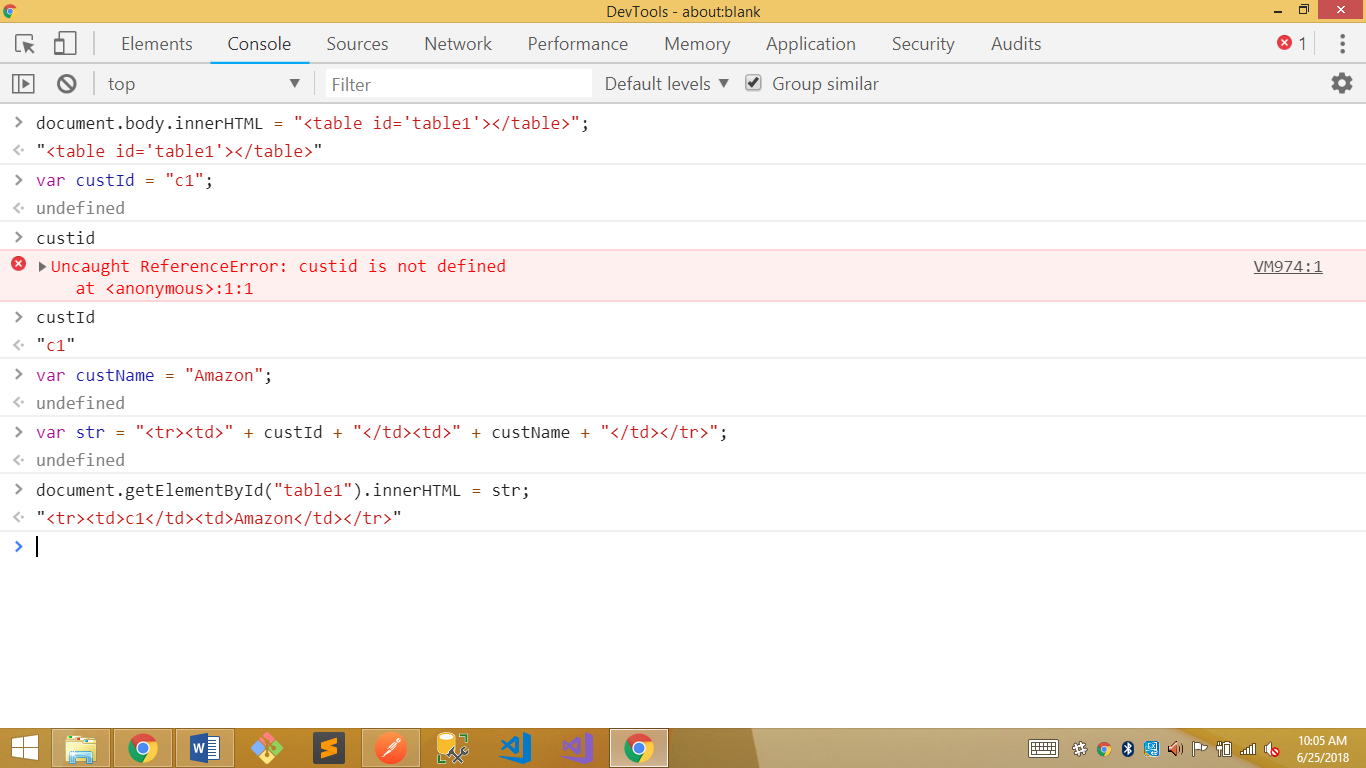
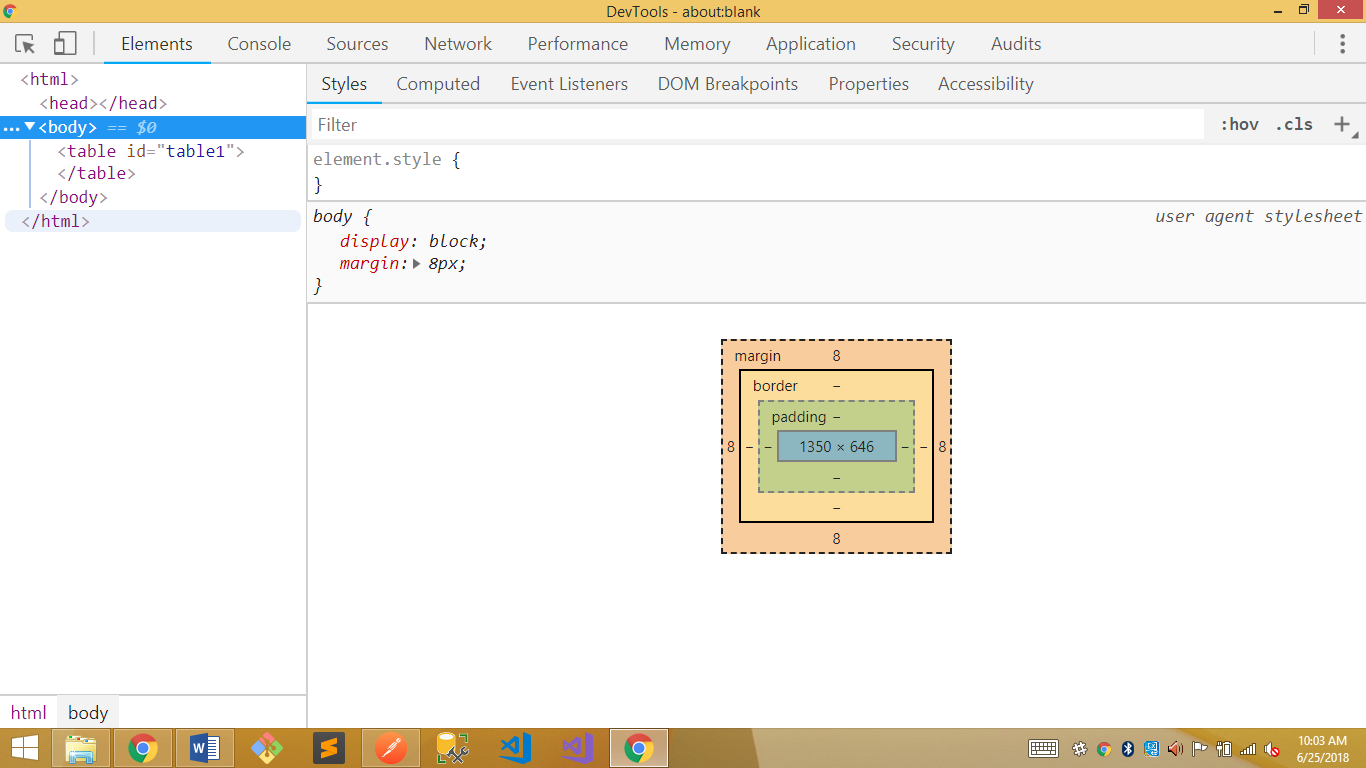
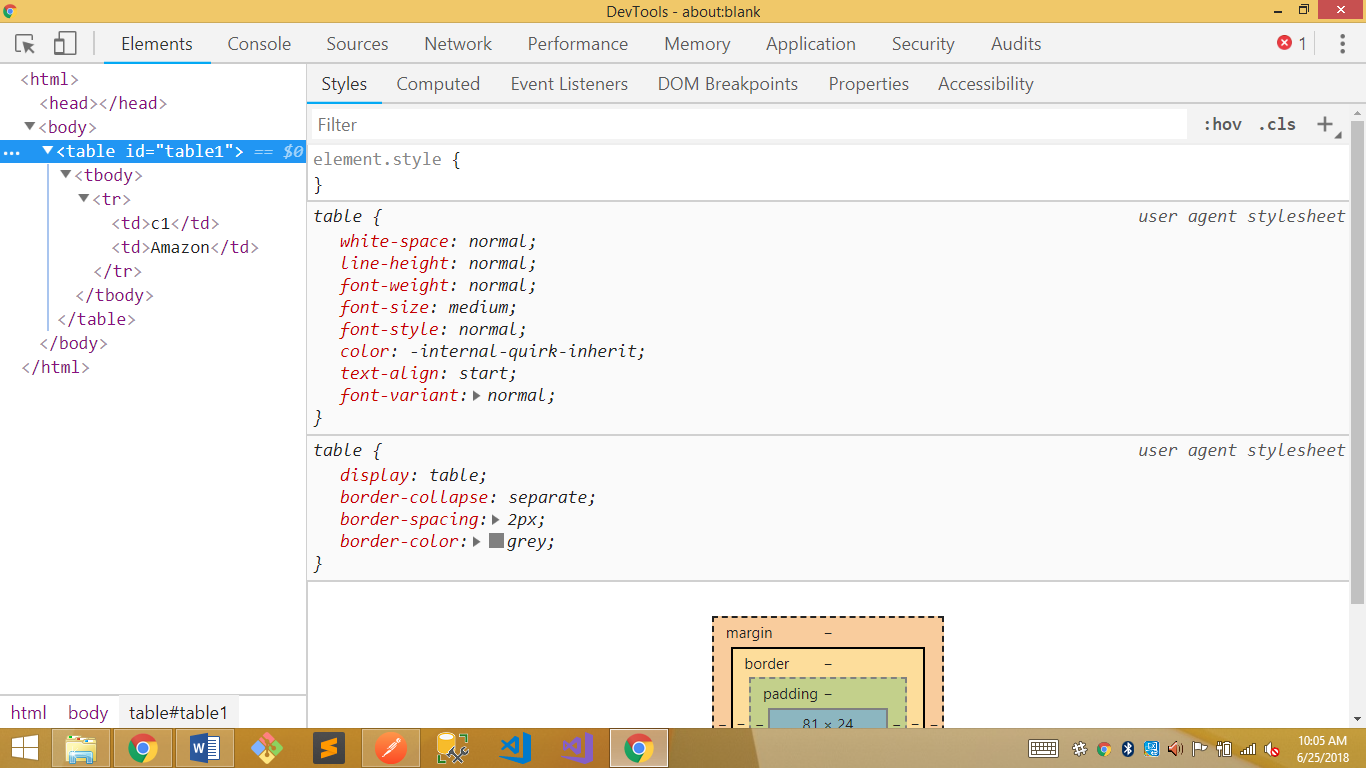
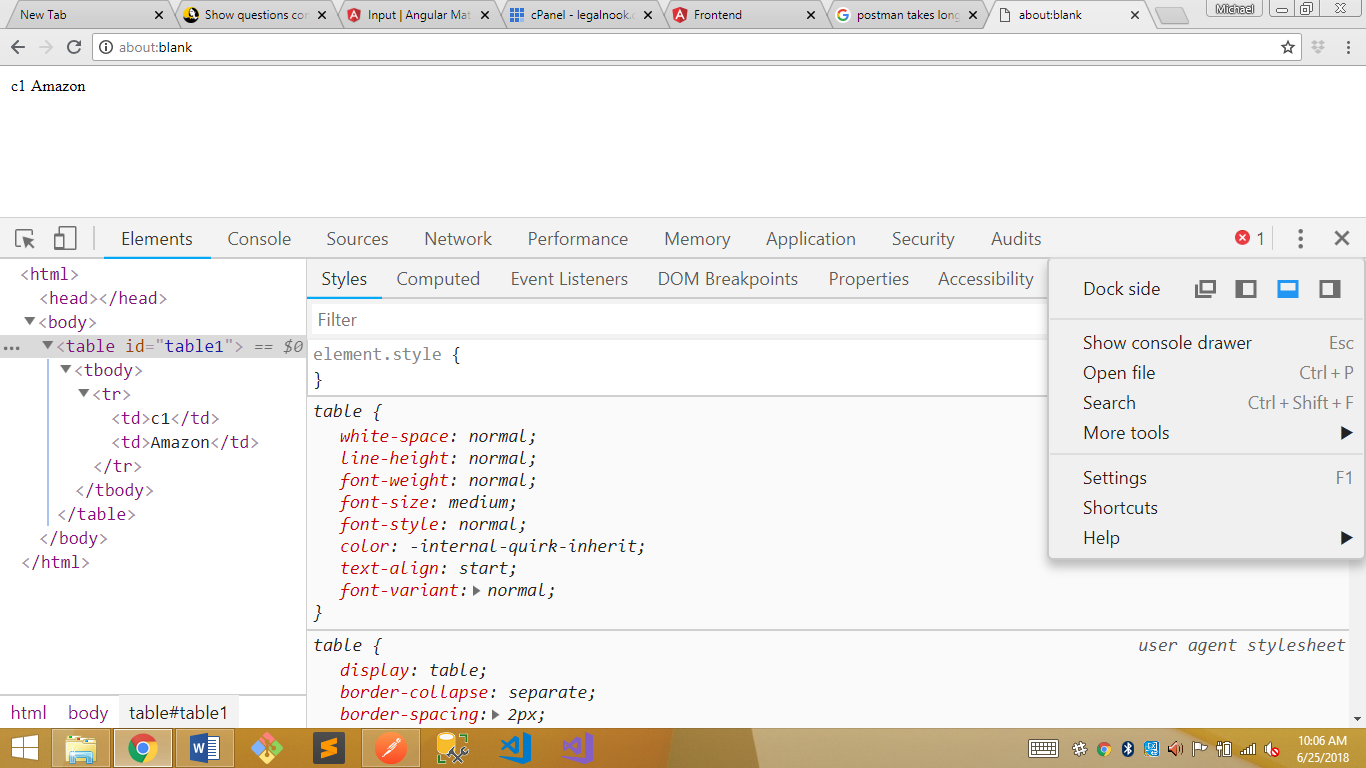
* In JavaScript we only call the method name like abc() and not the instantiation and then method name like in C# (e.g., t.abc();)
* There is a foreach-like function in JavaScript but it is done a little differently – Greg said we will get to this later

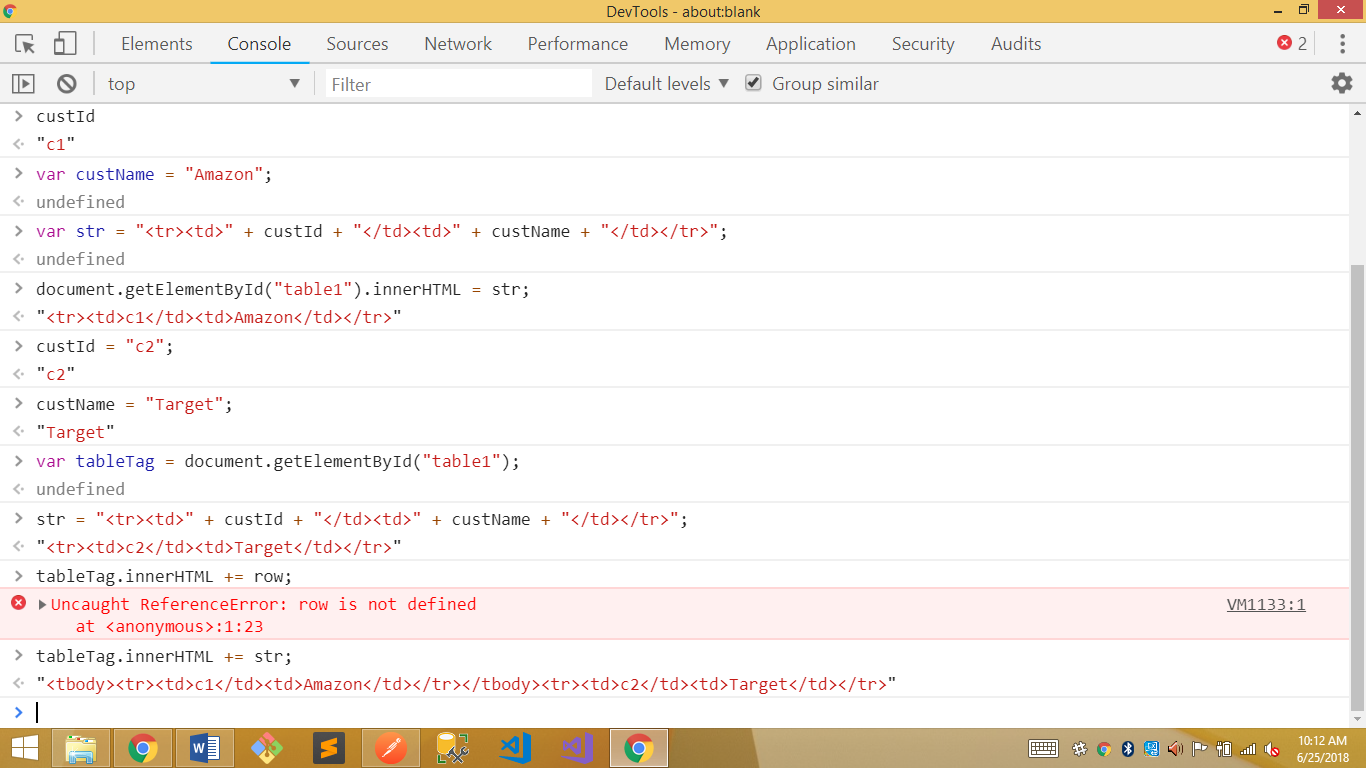
***6/25/2018 (Day 25):***

***Document Object Model:***

* Module 2 of the JavaScript section of our workbook regarding the Document Object Model (“DOM”)
* These typically start with the “document.”
  + Then document.firstChild which is the first html tag in your page (i.e., the HTML tag itself)
  + This can be done down through the child of child and so on
* We use these by (for instance) looping through the children, getting a div and getting all the attributes in that div so we can programmatically alter things within
* document.firstChild.childNodes would give us the head/body tag in an array
  + And document.firstChild.childNodes[1] would give us the body tag
  + And document.firstChild.childNodes[1].childNodes would give us all the body tags
  + Usually just use document.body rather than the above to get the body
* We use document methods to find elements
* Three ways to style are (1) class, (2) Id, and (3) tag
* Greg stated he uses getElementById() more than any others when searching for elements (pg.23)
  + Once we get the Id we can set properties on that HTML tag
  + Typically this is for form controls such as input data
* var tagCtrl = document.getElementById(“[Idname]”);
  + Greg uses “var tagCtrl” by convention
  + So…tagCtrl.value allows you to access the value property and it will give you what they keyed in
* If you try to call a property on a misspelled div then it will return null and your program may fail
* Greg stated he has never used searching for elements using CSS selectors
* We will use “onclick“ more than any other of the common events (pg.25)
  + onclick events will usually call a JavaScript function that you created to do something
* we skipped event bubbling/event propagation --- we are going to stop this from occurring on our website --- Greg said to review it once we learn this well
* Adding elements to the DOM -- we will be doing this often in our Capstone Project (pg.28)
  + We build rows to put in tables dynamically from json packages
  + we practiced this in blank chrome tab in developer mode:





* we can also create new elements via document.createElement(“[element name]”) such as document.createElement(“table”) (pg.29)
  + you can then style as follows:
    - var newTable = document.createElement(“table”);
    - newTable.border.color = “red”; //changes border of table color to red
* adding attributes is done similarly as above for adding elements (pg.29)
* removing things (pgs.31-32)
* delete in reverse order (pgs.33-34)

***Module 5: Diving Deeper (pgs.59-86):***

* JavaScript is not compiled
* eval() function lets you dynamically interpret and Greg suggested we never use this because it’s hard to debut
* Json.stringify turns the json into a valid string ---probably won’t have to use this ourselves but will see it with other technologies
* console.warn shows up in different color than console.log
  + usually we are the only one that sees these and are generally used while we program
* navigator properties (pg.64) are used to see what version of the browser will be running your code
* Number variables (pg.68)
  + toLocaleString(locale,options) good for formatting money (i.e., dollar signs)
* useful math functions:
  + ceil
  + pow
  + random
  + round
  + min
* escape characters (pg.69)
* String methods (pg.70) – Greg said to make not of these because they are valuable when you have to parse strings in a variety of ways
* Global scope (pg.73)
  + We may run across code that has this
  + Global scope means it is accessible from anywhere – they are defined outside of any function with or without var or defined within a function without using var
  + If you add “use strict” to a scope then attempts to create variables without declaring them with “var” will cause an error – we will use this at top of our Javascript files – covered in detail on pg.74
  + Greg said don’t use global variables as they cause a lot of trouble
* Anonymous functions(pg.77)
  + Anonymous function is simply a function without a name
  + “var x = function() {}”(var function) is the same as “function x() {}”(named function) and we would call both with “x()”
    - Can only be used inline and not called with something else
    - Different between the two above is if created at a function scope, the name function is “hoisted” to the Global scope while the function assigned using “var” is at the function scope
  + These are functions in every way but only gets executed when an event of some kind happens
    - e.g., mybutton.onclick=function() {alert(“you clicked me”)};
* The “this” keyword (pg.80)
  + Refers to owner of the function where the keyword is used or, if not a function, then it refers to the window object
  + Most common use of “this” is to pass an element to a function
* Greg said to use Module 5 as a reference as this is a tough chapter

***Visual Studio Code (JavaScript):***

* We then practiced some JavaScript in Visual Studio Code
  + Filename: button-click.html
  + Filename: button-click.js
* Minified version helps make your code hard to read so people can’t copy it and reuse it easily (e.g., bootstrap.js vs bootstrap.min.js) – downside of using minified version are unreadable or incomprehensible error messages

***Tables:***

* Table has 3 high level tags: <theader>,<tbody>, and <tfoot> which we can use --- Greg said only one he ever uses in <tbody>:

<table>

<tr>------</tr>

<tbody id=”tbody”>

<tr>-------</tr>

</table>