**Monday 2/19**

**TESTING AND TEST DRIVEN DEVELOPMENT**

1. Test-driven dev (TDD) is a software dev process that relies on the repetition of a very short dev cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the min amount of code to pass that test, and finally refactors the new code to acceptable standards.
2. PROCESS
   1. Add a test
   2. Run all tests and see if the new one fails
   3. Write some code
   4. Run tests
   5. Refactor code
   6. Repeat
3. WRITING TESTS
   1. Tests should include:
      1. An ID for the test to uniquely id it
      2. A description for what it is testing
      3. The test itself
      4. The expected result of the test
      5. The actual result of the test
   2. A test passes if Expected is the same as Actual even if an error is expected
   3. Tests can be grouped in categories
4. EXAMPLE
   1. TESTING: int Divide(int num, int denom)
      1. ID DESCRIPTION EXP ACT PASS
5. RPN Calculator

* Reverse Polish Notation
* Uses a stack data structure
  + LIFO
* Standard: (1+2)\*3=9
* RPN: 1 2 + 3 \*
* Can use Stack<> class

Example:

(1+2)\*(3+4)

In RPN – 1 2 + 3 4 + \*

**Tuesday 2/20**

1. Visual Studio has a Test Explorer providing automated testing for C#
2. Add testing to a program by:

* Add a test type project to solution
* Add reference to test project
* Add using for namespace
* Add classes with [TestClass] attribute to class
* Add methods with [TestMethod] attribute to class
* Add ‘Assert…’ statements

1. Run test from TestExplorer

* Test > Run > All Tests

1. ASSERT

* Statement within a test method defining the individual test to be run
* Many variant are available:
  + AreEqual, AreNotEqual
  + AreSame, AreNotSame
  + IsTrue, IsFalse
  + IsInstanceOfType, IsNotInstanceOfType
  + IsNull, IsNotNull
  + ThrowsException

1. Running Tests

* Click ‘Run All’ or one of the run options at the top of the test Explorer window
* Or in the menu: test > Run > All tests
* Results

**Wednesday 2/21**

ENTITY FRAMEWORK

1. Handout given
2. Anything with “core” (EntityFramework Core) is talking about tech that also works in non-Microsoft environments. Subset of total EF.

**Thursday 2/22**

ENTITY FRAMEWORK – MODEL ATTRIBUTES

1. Nullable Numbers

* There are reasons to allow scalar in C# to be null in the database
  + Int, long
  + Boolean
  + DateTime
* A Foreign key of type int that can be null
* Use the ? along with the scalar type
  + Public int? CustomerId {get; set;}

1. Why model attributes?
   * + Model attributes bridge the gap between classes and properties in C# and SQL types
     + Examples:
       - String in .net have a virtually unlimited length. SQL requires a max string length
       - A SQL column name needing a diff value than the associated property name in C#
     + Multiple attributes may be added to the same object.
2. Key

* Used to ID the property or properties that make up the primary key
* EF will assume a property is a primary key if it uses the name Id or concatenates the class name with Id (e.g. StudentId).
* A property StudentKey would need this attribute
  + 1. [Key]
       1. Public int StudentKey {get; set;}

1. Required
   * Used to id that a value is always needed for this property
   * Causes EF to generate the field with not null
     1. [Required]
        1. Public string Username {get; set;}
2. MaxLength(n)/MinLength(n)
   * Used todefine the max and the min length oif string properties
   * MaxLength is used to set size on nvarchar fields
   * An optional error message can be defined

[MaxLength(30, ErrorMessage=”Max length is 30”]

public stringUsername {get; set:}

1. NotMapped
   * Used to indicate that this property should not be created or maintained in the table
   * Typically used for derived properties
     1. [NotMapped]

[Table{“Member”)]

Public class Employee {…}

1. [Table([name])
   1. Defines the name of the table in SQL associated with the class
   2. Use when the table in SQL should be diff than the class name.

[Table([Member])

Public class Employee

1. [Column([name])]
   1. Defines the name of the column in SQL associated with this property
   2. Use when the column in SQL should be diff than the property name

Column([“ZipCode”])]

Public string PostalCode {…}

1. [DatabaseGenerated]
   1. Used to indicate that this property is generated by the database so EF should not add or update this even if supplied. It is always returned on read operations.

[DatabaseGenerated DatabaseGenerated=computed)]

Public DateTime DateCreated {get;set;}

* 1. Rarely used

10. [Index([name], IsUnique=true)]

* + - Use to cause an index to be created on the column in the database. The index is unique if IsUnique = true is added.
      1. For uniqueness on a non-key column
      2. For fast searching by a column
    - Without [name], a cryptic index name is generated.
    - The name parameter is almost never used.

[Index (IsUnique=true)]

Public string Username {get;set;}

**Friday 2/23**

Lab Day

**Monday 2/26**

Front End Begins: HTML, JavaScript, CSS

HTML

1. HTML
   1. The core tech for the web
   2. A *markup* language based on a hierarchy of XML tags
   3. Used to organize data on a web page
   4. All tags have a default style that can be modified by CSS
2. TAGS
   1. Names surrounded by <> (e.g. <html>)
   2. Most but not all have opening and closing tags (e.g. <p>.. </p>)
   3. May contain attributes (e.g. <div id=”abc”>)
3. <!doctype html>
   1. This is a preamble (directive) that provides info to browsers
   2. Makes sure browsers render the document based on the specification standards
   3. Should always be the first line in every HTML document
4. <html lang=”en”>
   1. The root of an HTML document
   2. All other tags must exist within the HTML tags
   3. May contain only one <head></head> followed by one <body></body>
5. <head>
   1. Contains metadata about the document
   2. Must contain the <title> tag
   3. May include information about:
      * Author
      * Description
      * Css
      * javascript
6. <body>
   1. Contains the visible elements for the html document
   2. There can be only one body tag
7. Inline Elements
   1. Inline elements are rendered to flow right along with any other elements
   2. No space is added above, below, right or left
   3. Examples of inline elements
      * <a>
      * <button>
      * <img>
      * <input>
      * <select>, <option>
      * <span>
8. <a>
   1. Anchor; sometimes called a link
   2. Renders a string that, when clicked, transfers to another URL
   3. Requires HREF attribute
   4. Content is the text to display on the page
      1. <a href=”http://go.com” >Go Dot Com</a>
9. <button>
   1. Button
   2. Renders a button containing text
   3. Typically used with OnClick event
      1. <button>Click Me!</button>
10. <img>
    1. Image
    2. Renders and image
    3. Image is specified with src attribute
    4. Size can be modified with height and width
       1. <img src=”logo.png” height=”200” width=”200”>
11. <input>
    1. Input – a form tag
    2. Used mostly to render an input box
    3. Can render a number of diff form elements based on the type attribute
       1. <input type=”text”>
12. <select>, <option>
    1. Select from options
    2. Renders a list or dropdown list with a defined set of options
    3. Options are defined using <option> tags withing the <select> tags. Options can use the value attribute

<select>

<option> red

<option>blue

</select>

1. <span>
   1. Span – Group collection of data and elements
   2. Provides the ability to apply a style to a group of data and/or elements
      1. <span> Treat this text as a group</span>
2. **<block>**
3. <div>
   1. Div- Group elements
   2. Sim to a span tag, but div is a block element
      1. <div>

<p>Some cool text! </p>

<img src=”logo.png”>

</div

1. <form>
   1. Form – Collection of user input tags
   2. Attribute method = get or post
   3. Attribute action = URL
   4. May contain a button with type submit

<form method=”get” action=”..”>

<input type=”type”>

<button>Clickme</button>

</form>

1. <h1>..<h6>
   1. Heading tags 1 through 6
   2. Renders text in larger and bolder style (h1) to smaller and less bold (h6)
      1. <h1>
2. <ol>,<ul>,<li>
   1. Ordered and unordered lists
   2. Creates list with either bullets (<ul>) or numbers (<ol>)
   3. Items in the list are declared with <li> tags within the <ol> or <ul> tags

<ul>

<li>Red

<li>blue

</ul>

1. <p>
   1. Paragraph
   2. Renders contents inline with a blank line above and below

<p> This text will have a blank line above and below it</p

1. <table>,<thread>,<tbody>,<tr>,<td>
   1. Table
   2. Renders text in rows and columns
   3. Thread – defines the table headers
   4. Tbody – defines the table data
   5. Tr – defines each row with one or more columns
   6. Td - defines one column in one row

<table>

<tr><td>Label</td><td>Data</td></tr>

</table>

1. Attributes
   1. Attributes add more info or identification to an html tag
   2. Values for attributes should be enclosed in quotes
   3. Common attributes:
      1. Id
      2. Style
      3. Class
      4. Value
2. Id attribute
   1. Provides a unique identifier for any tag it is applied to
   2. The identifier must be unique among all other identifiers on this html block document
   3. Very useful to programmatically access a particular tab
      1. <div id=”top”>
3. Style attribute
   1. Provides the ability to attach CSS styles to a tag

<p style=”color:red;”>

1. Class attribute
   1. Provides the ability to assign a CSS class style to a tag
   2. The class style must be defined in an included stylesheet
   3. Multiple class styles are separated by a space and are applied in order of appearance

<table class=”table tbl-small”>

1. Value attribute
   1. Provides the ability to get and set the contents of a form tag.

<input value=”ABC”>

1. Events
   1. Events are ways to respond to user interactions with the html page
   2. To respond to an event, the event must be added to the tag along with the code to execute when the event occurs
   3. Common events:
      1. Onclick
      2. OnLoad

CASCADING STYLE SHEETS

1. Borders
   1. Bounding box outside the text content
   2. Multiple ways to specify the box
      1. Border: 1px // all 4 sides

:1px 2px; //T&B, L&R

:1px 2px 3px 4px; TBLR

Border: 1px solid black;

Border-top|border-right|border-bottom|border-left

1. Margin
   1. Spacing outside the borders
   2. Usually expressed in px or em
2. Padding
   1. Spacing inside the borders

**Tuesday 2/27**

1. Stylesheets
   1. Styles can be applied externally to the style attribute by specifying:
      1. Tagname
      2. Id
      3. Class
   2. Styles can be defined in the <style> tag of the page or in a separate file and included using the <link> tag
   3. <link rel=”stylesheet” type=”text/css” href=”theme.css”>. Link is useful when you have many pages with many diff styles
2. Styles by tagname
   1. Includes the tag to apply the style to
   2. Applies the style to EVERY tag in the page
   3. Style definition may contain multiple styles
3. Styles by id
   1. Must include the id of the tag to style
   2. Id value must be start with a hash char (#)
   3. Applied to the tag (if any) with the id value
      1. #main{
      2. }
4. Styles by classname
   1. Class name can be any unique name
   2. Name must begin with a period
   3. Applied to any tag using the classname in the class attribute
5. Styles in a stylesheet
   1. Styles can be defined in a file (.css) and include in a page using the <link> tag
   2. Multiple stylesheets can be included in the same page.
   3. Last style encountered wins!
      1. <head>

<link rel=”stylesheet” Type=”text/css” href=”master.css”>

<link rel=”stylesheet” type=”text/css” href=”age.css”>

1. Multiple applications of the same style
   1. The same style can be applied to the same tag, id, or classname
   2. Separate diff tag, id, or classname by a comma before style definition
      1. Table, tr, td, .smaller,
2. Selectors
   1. Defining styles by tag, id, or classname are just 3 specific selectors for defining how styles are applied
   2. A more complete list is at w3schools.com/cssref/css\_selectors.asp
3. Selector tag inside tag
   1. Selector specifies to style all tags that are contained inside another tag
   2. Tags are separated by a space
   3. /\*paragraphs inside a div \*/
      1. Div p{
   4. Color: red: }
4. Selector tag immediately inside tag
   1. Selector specifies to style all tags that are the first tag contained inside another tag
   2. Tags are separated by a plus (+)
      * 1. Div + p { color: red;}

**Wednesday 2/28**

1. JavaScript
   1. The programming language of the browser
   2. The language is sim in syntax to C# and Java
   3. Interpreted; not compiled
   4. Dynamically typed. C# is statically typed
2. Syntax
   1. Case Sensitive
   2. Statements *should* end with semicolon. C# requires it. Best practice to just use it in both places
   3. Exists within <script> tags
      1. Should include the lang=”javascript” attribute within the html page
3. Types
   1. JavaScript has five basic types:
      1. Number
      2. String
      3. Boolean
      4. Array
      5. Object
   2. Other types:
      1. Null
      2. Undefined
   3. Can use typeof n to get the type of a variable

Var I = 0;

Typeof I; //returns “number”

1. Dynamically Typed
   1. All variables are defined using var
   2. Type of variable is based on the data stored in the variable:
      1. Var n = 1;
      2. Var n = “1”;
      3. Var n = true;
      4. Var n = [1,2,3];
      5. Var n = {id: 1}; //type object
      6. Var n = null;
      7. Var n; //undefined
   3. Once initialized, a variable can be set with any other type.
2. Number
   1. Any type of number:
      1. Integer
      2. Decimal
      3. Engineering
      4. Var m = 1;
      5. Var n = 2.3
      6. Var p = 2.5e3; // 2,500
3. Arithmetic operations
   1. Increment(++): var z = x++ //z=4,x=5
      1. Var a = 10;
      2. A++ //A=A+1;
      3. Var b = a++;
      4. Var c = ++a;
      5. It makes a diff where the ++ is. In front of – before, behind, after.
4. String
   1. Collection of characters surrounded by quotes (single or double). Use double quotes as a rule.
      1. Var n = “abc”;
      2. Var m = “O’Reilly”;
      3. Var p = 1.toString();
5. String Functions
   1. toString()
      1. var s = 1.toString(); // s = “1”;
   2. length
      1. var len = “abc”.length; // len = 3
   3. indexOf(..)
      1. var pos – “abcde”.indexOf()”cd”); // pos = 2
   4. substr(..,..)
      1. var str = “abcde”.substr(1,2); //str = “bc”
   5. toUpperCase()/toLowerCase()
      1. var ustr = “abcde”.toUpperCase(); //ustr = “ABCDE”
6. Boolean
   1. Possible values are true or false
      1. Var n = true;
      2. Var n = (a == b);
      3. Var n = !n;
7. Array
   1. A collection of JavaScript data of any type
   2. Items are separated by commas and enclosed within square brackets e.g. [1,2,3]
8. Array functions
   1. Arrays are indexed starting at zero
   2. Length: returns the number of items in the array
   3. Join(ch): curates a string from the array items with each item separated by ch
   4. Push(x), pop()
9. Object
   1. Collection of key/value pairs
      1. Var obj = { id: 0, name: “Lisa”};
   2. Bounded by curley braces
   3. Key is a string WITHOUT QUOTES
   4. Value is normal JavaScript Type
   5. Separated by a colon
   6. Multiple key/value pairs are separated by a comma
10. Null
    1. The absence of a value
    2. Different than undefined
11. Undefined
    1. Variable type when declared but before a value is assigned
       1. Example – var n; //type is undefined at this point
12. Typeof n
    1. Used to return the type of a variable
    2. To check if available is an array, use:
13. Functions
    1. Similar to methods in C# and java
    2. Name must be unique
    3. May take parameters (e.g. message)
    4. May return data
    5. Executed by name and parentheses
       1. (Debug(“a”))
    6. Example:

Function Debug (message) {

Console.log(message);

Return true;

}

1. Boolean Expression
   1. Any logical expression that evaluates to a boolean value
   2. Typically involves comparison and/or logical operators
   3. May use parentheses to alter precedence
2. Statement Block
   1. A statement block is one or more statements surrounded by curly braces
   2. Can be placed anywhere a single statement is defined.
      1. Example

If (Boolean-expression)

Statement | statement-block

1. Assignment
   1. A single equal sign represents assignment
   2. Evaluate everything on the right of the equal sign down to a single variable value. Place that value into the variable on the left of the equal sign.
2. Equality
   1. 1 == “1” evaluates true in JavaScript
   2. 1=== “1” evaluates false. Triple Equal means value and type must equal. Much like == in C#.
   3. == - two objects are equal if they have the same or compatible value
   4. === - two objects are equal if they have the same value and type.
3. If
   1. Execute a statement block only if a Boolean expression is true
   2. Optional else clause
4. While
   1. Execute a statement block repeatedly as long as the Boolean expression is true.
5. For (indexed)
   1. Loops through a set of statements modifying a variable after every iteration
6. Ternary operator
   1. Used to assign one of two values to a variable based on some Boolean expression
7. Math.floor(Math.random() \* 11)
   1. This will give a random number between 0 and 10
8. JavaScript assignment:
   1. Bowling is a game of 10 frames. The random number generator can simulate the number of pins knocked down. Must do 10 times to constitute 1 game.
   2. Create an HTML page (bowling.html)
   3. Add JavaScript code (<script lang=”javascript”>) //in head section
   4. Add a java Script function call PlayBowling()
   5. Inside the Playbowling function:
      1. Create a variable of type array and init it as empty
      2. Create a for loop to iterate 10 times, once for each frame.
      3. Inside the for loop :
         1. Get a random number between 0 and 10 inclusive
         2. Store the random number in the array
      4. When the for loop is done
         1. Turn the array into a string using Join(“,”)
         2. Var arr = [1,2,3];
         3. Arr.join(“,”);
         4. “1,2,3”
         5. Var str=arr.join(“,”);
         6. Console.log(str);
         7. 1,2,3
   6. Without jquery
      1. On the <body> tag, add the OnLoad event
      2. <body onLoad =”Playbowling()”> (developer mode open)

**Thursday 3/1**

1. Switch
   1. Like case statement
   2. Executes a block of code based on the value of a variable
2. Document.getElementById()
   1. Gets a reference to a particular html tag by specifying the value of the id attribute
   2. Returns only a single item
3. Document.getElementsByTagName()
   1. Gets a collection of ALL tags with the tag name
      1. Var ctrls = document.getElementsByTagName(“p”)
4. Document.write|writeLn
   1. Writes text to the body of the document
      1. Document.writeLn(“Some text written”);
5. Console
   1. Operations using the browser console
      * Clear() – clears the console
      * Error() – writes error message in red
      * Info() – writes info message
      * Log() – writes log message
      * Warn() – writes warning message
        + Conole.log(“text to write”)
6. Elements
   1. Properties of the DOM
      * GetAttribute() – gets an attribute value of a tag
      * Click() – programmatically ‘clicks’ the tag
      * innerHTML – full HTML contained within tag
      * parentElement – the parent of a tag
      * textContent – get or set a tag’s content
7. Events
   1. OnClick- called when a tag is clicked
   2. OnLoad – called when the tag is completely loaded
   3. OnChange – called when <input> or <select> value is changed
   4. OnSubmit – called when the form is submitted.
8. Window
   1. Highest level object
   2. Includes document, console, and others
   3. Height and width of the window

**Friday 3/2**

1. JavaScript Practice Day

**SIMPLE ARRAY SUM IN JAVASCRIPT FROM HackerRank**

process.stdin.resume();

process.stdin.setEncoding('ascii');

var input\_stdin = "";

var input\_stdin\_array = "";

var input\_currentline = 0;

process.stdin.on('data', function (data) {

input\_stdin += data;

});

process.stdin.on('end', function () {

input\_stdin\_array = input\_stdin.split("\n");

main();

});

function readLine() {

return input\_stdin\_array[input\_currentline++];

}

/////////////// ignore above this line ////////////////////

function simpleArraySum(n, ar) {

// Complete this function

var total = 0;

for(var item of ar){

total += item;

}

return total;

}

function main() {

var n = parseInt(readLine());

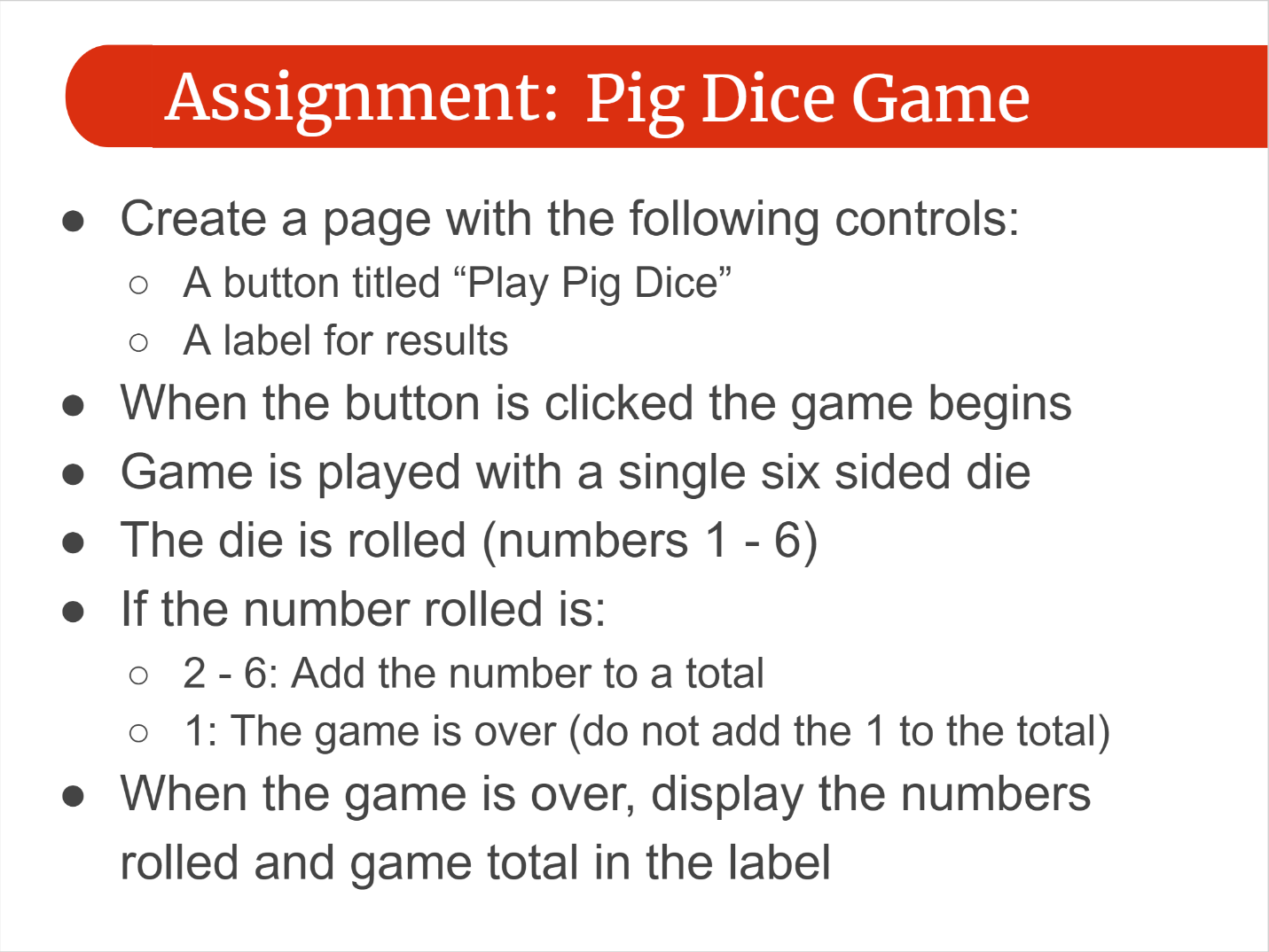
ar = readLine().split(' ');

ar = ar.map(Number);

var result = simpleArraySum(n, ar);

process.stdout.write("" + result + "\n");

}



Need this function:

Math.floor(Math.random()\*6) + 1 [1-6]

**Monday 3/5**

1. AJAX
   1. IP address - 192.168.1.10
   2. Domain Name Server (DNS) - Maxtrain.com
   3. DNS translates to unique IP
   4. Browser only understands HTML, CSS, and javascript
   5. JSON – JavaScript Object Notation
   6. How the client communicates to the server

**Tuesday 3/6**

1. MVC
   1. Model/View/Controller
   2. Model: User class
   3. View: Visual representation of the model
   4. Controller: list function, one controller for each model
      * + Example:

User(class) – Model

UsersController – Controller

List() – method in Controller

Get(int id) – method in Controller

* 1. URL: http://localhost:12345/Users/Get/1

Users is controller

Get is method

**Thursday 3/8**

1. MVC

**Wednesday 3/14**

Angular/Typescript

1. Prerequisites
   * 1. The following technologies are prereq’s to learning Angular:

JavaScript (esp Json, arrays, objects)

HTML

CSS

Bootstrap

* + 1. Typescript is a subset of Javascript

1. Why use a JavaScript Framework??
   1. Angular is an application framework written in Typescript for building responsive, web application and Single Page Applications. It provides all the tools and libraries needed. It has gained wide acceptance in the marketplace.
   2. A framework is a toolset for producing applications that requires the developer to follow a set of guidelines on how to create applications
   3. Angular is a Google product
2. Angular and AngularJS are NOT the same thing.
3. Google Search
   * 1. Angular Search “-AngularJS”
4. Typescript
   1. Superset of Javascript created by Microsoft
   2. Provides stronger type checking in development mode for:
      1. Properties (variables)
      2. Method Parameters
      3. Method Return Values
   3. Is transpiled (compiled) into JavaScript
   4. Example:

Var a;

A=1;

A=”2”;

Var b=a+1;

B=”21”

1. Installing NodeJs
   1. Installing Typescript requires using NPM
   2. Install the Long Term Support(LTS) version of NodeJs at <https://nodejs.org/en/>
2. Installing Typescript
   1. To install typescript using NPM ( -g is global):
      1. >npm install –g Typescript
      2. Tsc –v ( use to see if typescript compiler (tsc is installed, i.e, typescript is installed)
3. Transpiling (Compiling)
   1. Because the browsers of today do not understand Typescript, it must be turned into JavaScript
   2. This is the job of the Typescript program tsc
   3. The Typescript compiler turns a Typescript file (\*.ts) into a JavaScript file (\*.js)
   4. It can be done on file at a time:
      1. tsc HelloWorld.ts creates HelloWorld.js
   5. Or it can watch an entire folder:
      1. tsc –w
   6. Any JavaScript is ignored
4. Comparison of Typescript and JavaScript

Typescript vs. JavaScript

Let a: number = 0; var a = 0;

Var b = 7; var b = 7;

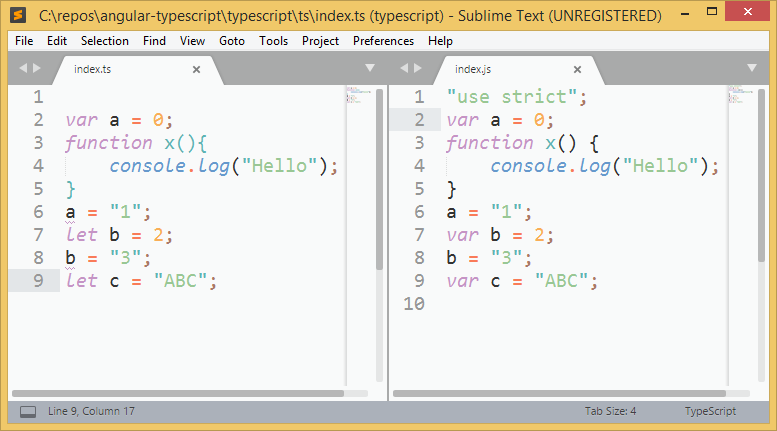
Function Add(p1: function Add (p1, p2)

number, p2: number): {

number{ return p1 + p2;

Return p1+ p2; }

}

1. Tsconfig.json
   1. The typescript config file
   2. Created with tsc –init
   3. Key config options:
      1. Outdir: put \*.js files in this directory
      2. Baseurl: base directory to resolve modules name
      3. Paths: object of symbolic names @models/\*:app/models/\*
      4. “include”:[“./\*.ts”]
   4. Must exist in the root directory of the project
2. Editor plugins
   1. Many programmer editors have
3. 
4. Types:
   1. In Typescript, types are declared by appending a colon “:”to the name of the object (variable or function) and specifying
   2. Number
   3. String
   4. Boolean
   5. Array
   6. Object : collection of key: object pairs
   7. Tuple : fixed array
   8. Enum : friendly name to numeric value
   9. Any : declares any other type is valid
   10. Void : does not return a value
   11. Null : the absence of a value
   12. Undefined : value has never been assigned
5. String
   1. Surrounded by single or double quotes as in JavaScript
   2. Backtick ` char can surround string spanning multiple lines
   3. Can use interpolation with ${a} within string
6. Type Assertions
   1. Indicates the use of a value as a diff type

Let a = (1 as string); // <string> 1

Typeof a ; // returns “string”

1. Let
   1. Is used to define vaiables using block scoping
   2. Should always use let versus var.
2. Const
   1. Defines a variable as having an initial value that cannot be changed.
      1. Const url = “http://gmail.com”; (use in project where we have localhost://
3. Interface
   1. Defines a structural layout required of anything implementing the interface
   2. In the following interface, reversed is an optional properties in objects implementing IPrintable
4. Class
   1. Defines objects that bind the properties and the functions that operate on those properties
   2. May have only a single constructor

Class Square {

Side: number;

Constructor (side: number) {

This.side = side;}

Area(): number { return this.side \* this.side;}

}

**Friday 3/16**

Git and Branching

1. Branch makes a copy of current repos state of master
2. Your work gets done in the branch, the master stays the same
3. To get your changes to master, you issue a merge command
4. When working in group, id your branch by it’s functionality.
5. Include your name in the branch
6. Someone has to manage the branches that are pushed up “the owner of the project”
7. The developer pushes up to github but the “owner” does the merge
8. How to do branch:
   1. All in your Git
   2. Be in the branch you want to branch from
   3. Git branch <name> {this creates the branch}
   4. Git checkout <name> {use this To switch branches }
9. Merging
   1. Be in your branch you want to merge into (most likely the master)
   2. Git merge <branch name>
10. Delete the branch
    1. Git branch –d <branch name>

**Monday 3/19**

# ANGULAR

1. Angular is a framework for building web applications
2. Applications consist primarily of a hierarchy of components that are grouped into modules
3. A component and most Angular objects consist of 3 things:
   1. Class imports
   2. A decorator
   3. A class
4. When an application runs,
5. Placeholder
6. Single Page Application(SPA)
   1. Angular makes it easy to create an SPA
   2. An SPA has the following characteristics:
      1. Typically only a single HTML page with all the HTML tags common to every page
      2. HTML fragments (views) that get injected into the HTML page based on the browser URL
7. Command Line Interface (CLI)
8. The format to execute a command is:
   1. Ne [command] [ options] where command is required and options vary by command
   2. >ng new my-first-angular-project
9. Generate Component
   1. >ng generate component [option] [name]
   2. Most Angular projects have many components and all must be known to a module
   3. Creates a new subdirectory with the component name
   4. Generating a component not only creates the code for the component, but also imports it and makes it known to the associated module
10. Generate Module
    1. Ng generate module [option][name]
    2. A module is a grouping of components, services, pipes, and other modules
11. Generate Service
    1. Ng generate service [option][name]
    2. Generates a service and adds it to the module associated with it
    3. Example - Ng generate component menu
12. Build and run project (serve)
    1. Ng serve [option]
    2. Does a build and starts the project
    3. –o opens the browser or tab and monitors the source files refreshing the browser after any save
13. Also – start ng serve –o {to start another window for Ng to run in). Be in the directory of the project you want to start. (ex /c/repos/angular-typescript/angular/my-cv). Make sure another instance isn’t running in another git bash shell.
14. Importing Classes
    1. Virtually every file in Angular contains a class that has been exported
    2. To use one class in another class, the class must be imported which references:
       1. The file containing the class (user-list.component.ts)
       2. The classname itself (UserListComponent)
    3. File path is relative to path of the current file
15. Decorators
    1. Decorators provide meta-data about the object they proceed
    2. All decorators begin with an at sign
    3. Many decorators take a JSON object as a parameter
    4. Parameters vary by decorator
    5. Never add a semicolon to the end
    6. Classes are different types because of the decorators they use e.g. it is a component because it has a @Component decorator
    7. Most decorators take parameters, but unused parameters need not be specified
16. Classes
    1. Almost all objects in Angular are classes
    2. Classes must be exported using the export keyword
    3. Differences from C# and Java
       1. All objects are public by default
       2. Only one constructor allowed
17. Modules
    1. A module is an object in Angular that groups components, services, and other objects together
    2. All UI objects must belong to a module
    3. All applications have a root module
    4. All modules are marked with @NgModule decorator:
       1. Declarations: components
       2. Imports: modules
       3. Providers: services
       4. bootstrap: components started at run-time
       5. Exports: components available to import
    5. @NgModule
       1. Import {NgModule} from ‘@angular/core’
18. Components
19. Data Binding
    1. In angular, the data is bound to interface objects in either a one-way or two-way method
    2. In one-way binding, the modified property data value is immediately updated on the user interface, but changes to the user interface control does not cause a change in the property data.
    3. In two-way binding, any changes
20. \*ngIf
    1. Angular directive that controls whether the HTML it is attached and all it’s child tags are displayed or not based on a Boolean expression
    2. The asterisk is required
    3. <p \*ngIf=’display == true’>this is displayed only if the display property is true </p>
21. \*ngFor
    1. Angular directive that loops through a collection replicating the HTML tag and all child tags for each item in the collection
    2. Let is required
    3. <tr \*ngFfor=”let user of users”>

<td>Name</td><td>user.fnane</td>

<td>City</td><td>user.city</td>

</tr>

1. ngSwitch
   1. Angular directive that displays one of multiple sets of HTML tags based on a property
   2. <div [ngSwitch]=’abcValue’>

<p \*ngSwitchCase=’a’>Display a</p>

<p \*ngSwitchCase=’b’>Display b</p>

</div>

1. ngClass
   1. Sets the HTML class attribute using data from component
   2. <div [ngClass]=”classValues”>…</div>
2. Expression context
   1. Used to add or remove an attribute from HTML
      1. <p [hidden]=”!canview”>…</p>
      2. <button [disabled]=”isDisabled”>…</button>
      3. <img [src]=”imageUrl”>
      4. <div [class.special]=”isSpecial”>…</div>
      5. <p [style.color]=’color’>…</p>
3. Template reference variable
   1. Provides a programmatic way to access an HTML control (typically a form control)
   2. <input #firstname>
   3. <label>{{firstname.value}}</lable>
4. Events
   1. Defines functions to be executed when events on HTML tags fire
   2. Parentheses are required on event and function
   3. <button (click)=’log()’>…</button>
5. Component Communication
   1. Parent components can pass data to child component through the pseudo-html tags
   2. <vendor-comp [selectedVendor]=’aVendor’></vendor-comp>
6. Simples way to add Bootstrap to Angular is by adding the link to the src/index.html file
7. SERVICE Overview
   1. Services are objects that **can be shared across components**
   2. Must be imported and exposed in the providers decorator key of app module
   3. They are injected into components through the component constructor
   4. Common uses:
      1. Applications options
      2. User login information
      3. AJAX methods
8. Creating a service – ng generate service [servicename]
   1. Creates the service in the current directory
   2. Adds the service to the associated module
   3. Services are added to the providers key of the NgModule decorator
9. Example:
   1. Import {Injectable} from ‘@angular/core’;
   2. @Injectable()
   3. Export class SystemService{

loggedInUser: User = null;

}

* 1. Import { Component} from ‘@angular/core’;

Import {SystemService} from

‘./services/system.service’;

@Component()

Export class UserListComponent {

constructor(private SysSvc: SystemService)

{}

}

1. When a service is used to access data from a server, some rules to follow are:
   1. One service per server controller
   2. A class maps the data sent and received from the controller
   3. Minimal methods required are:
      1. Get all rows
      2. Get single row by primary key
      3. Insert
      4. Update
      5. Delete
2. HttpClient
   1. Service class within HttpClientModule providing Observable AJAX calls to server
   2. Must add HttpClientModule to AppModule
   3. Requires import of Observable from ‘rxjs/Observable’
   4. Injected into component via constructor
   5. Provides get and post methods
      1. Get for reading data
      2. Post for updating data
3. Observable
   1. Observables are constructs to handle asynchronous data
   2. Provided by importing from rxjs/Observable
   3. They are an upgrade from Promises (JavaScript)
   4. Observable provides subscribe method to handle data when returned

This.http.get(url)

* + - 1. .subscribe(res => console.log(res));

1. Example Ajax Service
   1. Import {Injectable } from ‘@angular/core’;
   2. Import { HttpClient } from ‘@angular/common/http’;
   3. Import {Observable} from ‘rxjs/Observable’;

//assuming a user class in user.ts

Import {User} from ‘./user’;

* 1. Get(id): Observable<User> {

Return this.http.geturl+’Get/’+id)

As observable<User>;

}

1. Routing
   1. Routing is the process where the URL is used as data for displaying the appropriate component
   2. Components are injected into HTML containing the pseudo-HTML tags
      1. <router-outlet></router-outlet>
      2. Special HTML tag to indicate where SPA routed components should be rendered.
   3. Requires a module to define the routes and the components associated with each route
   4. Order of routes **is significant**. More specific routes placed before more general
2. routerLink
   1. a special Angular attribute for <a> tags to programmatically link to a diff component by route.
   2. Used instead of href
      1. <a routerLink=”/about”>Go To About</a>
   3. Note: using href in <a> will display selected component, but will refresh the application which takes much longer, causes page to “blink”

**Monday 3/26**

* + - 1. In VS, new class, goes in Utility folder, ModelStateErrors.cs
      2. In Controller, in create, change and remove

If(!Modelatte.Isvalid) {

* + - 1. Var errorMessages = ModelStateErrors.GetModelStateErrors(ModelState):
      2. New Msg – 3rd variable , Data = errorMessages
      3. IN Msg.cs

Pick PR, additional action of Lines

That will give you PR and all it’s lines

Each line will have actions Create/Edit/Detail

We’ll have an authentication service that will be injected in all the components.