static void Main(string[] args)

{

/\*

\* The following can be seen as variables

\* Notice that they end with a semi-colon this is a must

\*/

//x;

//y;

/\* However these variables are missing something

\* Notice the red squally line under them and the red on the scroll bar

\* C# is a strongly typed language which mean at this variables

\* need a data type

\*/

/\* DataTypes tells the computer how much memory to set aside to hold the values

\* assigned to the variables, we will not go over how much memory at this point

\* DataTypes - string, int, double, decimal, char, bool((true/false) talk about later)

\* Let's redo and add variables

\*/

string x;

int y;

double z;

decimal a;

char b; // This can only hold one letter

/\* This is call declaring variables. Though they are declared and empty the

\* program will tell the computer to set aside memory for future use. The green

\* line under the variables just tells us that the variables have not been used

\*/

/\* You can declare more then on variable of the same DataType on one line. They

\* MUST be the same DataType

\*/

//string x, y, z;

/\* Notice that once again we have the red lines under the variables yet again.

\* This is because we have already used the variable’s names

\*/

/\* This brings us to how to naming the variables. The names that we have used so far

\* tell us nothing about what the variables would be used for. 'x' could be used

\* for anything from a name to a full sentence. So lets rename the 'x' variable

\*/

string firstname;

/\* While this does work it is still not correct. Notice that the it becomes harder

\* to read the variables when more then one work is used. To fix this we will use

\* what is called camelCase

\*/

string firstName;

/\* Notice that VS does not yell at use for using the same name. This is because C#

\* Does not see them as the same name because we cap one letter. There are a few other

\* rules for naming variables.

\* 1. The first character MUST be a letter or an Underscore (used later)

\* 2. After the first you can use letter, numbers, or underscore

\* 3. Name cannot have spaces

\*/

/\* Variables can only contain one value at a time. This bring us to how to read code

\* and how to read the '=' sign. For people not in programming this would be seen an

\* Equal sign. To programmers this is called the assignment operator.

\*/

firstName = "Thomas";

/\* The first thing to notice is that there is not DataType used. This is because when

\* we declared the variable, we already assigned a DataType. We could have assigned a

\* value to a variable when we declared the variables

\*

/\* You can assign value to a variable at the time of declaration \*/

string lastName = "Blake";

/\* To read the lastName line you would read it right to left. So this would be , "Blake"

\* is assigned (=) to a variable named lastName with a 'string' DataType

\*/

/\* There will be times when there will be variables that the values should never change

\* and stay constant. These will have different key word and naming convention. The Preferred

\* naming is all cap and underscore if more than one word.

\*/

//private const decimal STATE\_TAX = .07m;

/\* Notice the private access, we will go over those later and then the key word

\* const. This should stop the code from changing the variable later

\*/

/\* Global variables live outside any method. It is not a good idea to use Global

\* variables since all code will have access to the variable and can change it easily.

\* If we were to us the STATE\_TAX then we could safely use it a Global variable.

\*/

/\* Variables only live in the methods that they are declared in. This is called the

\* Scope of a variable. Later we will learn how we can use the variable’s value in

\* other methods.

\*/

/\* Concatenation is adding two string together. This can be done in several different ways

\* however, the use of '+' we must be carefull because that will add two number together

\* instead of printing then seperate.

\*/

Console.WriteLine("Thomas" + "Blake");

Console.WriteLine("First name " + firstName);

/\* Be careful \*/

Console.WriteLine(firstName + lastName); // This prints with no space between the names

// Another way

Console.WriteLine("His name is {0} {1}", firstName, lastName);

// One I like to use

Console.WriteLine($"His name is {firstName} {lastName}");

/\* Math in C# can be a challange. You MUST remember Order of Operation and program the

\* code to use the order. You can use number and variables mixed. Modules is one that

\* may not be known

\* % - return the remainder of two numbers

\*/

Console.WriteLine(9 - 1 \* 8);

Console.WriteLine((9 - 1) \* 8);

/\* Always remember to get the results you want you must us the (). Also if you are going

\* to assign a result of a calculation then you need to know the Datatype of the result.

\* If you the result of 2 int is a double and you assign it to an int variable the anything

\* after the decimal will be dropped (truncated). Decimals WILL NOT be assigned to a double

\* variable type.

\*/

Console.ReadLine();