DIGITAL DEVELOPMENT

Building a solution

AGENDA

INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

- Code Editor
- Simple Debugging Tools
- Compiler
- Error Diagnostics
- Run-time Environment (Console)
- Graphical User Interface

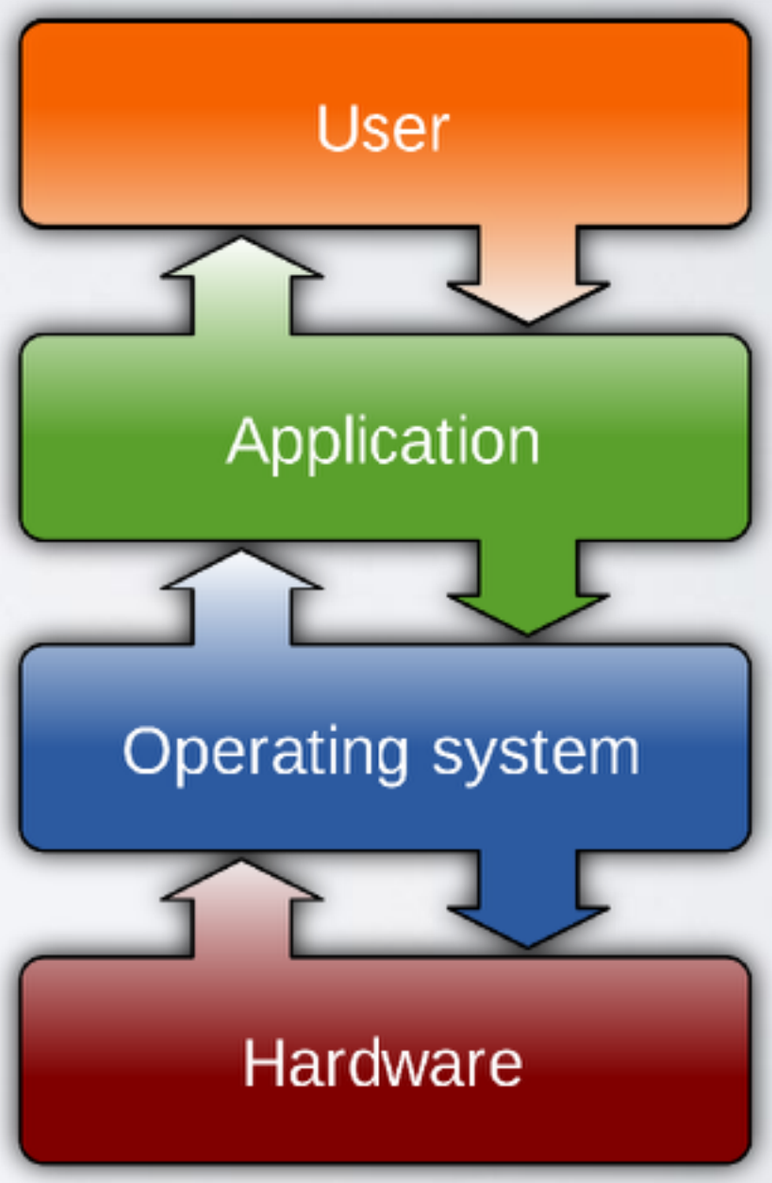
PROGRAMMING LANGUAGE FEATURES

- Data Types - Strings, Ints, DateTime, etc.
- Control Structures - If, If Else, Loops (While, For)
- Functions - Built In, User Defined
- Data Structures - Arrays
- String Handling - Concatenation
- Logical and Relational Operators (AND, <=, >=)

SOFTWARE

WHAT IS A SOFTWARE APPLICATION

- Instagram, Snapchat, Microsoft Word, Internet Explorer are all Software **Applications** that execute on Computer **Hardware**
- The application runs on top of an **Operating System** (Windows, OSX, Android, iOS) which simplifies how the Application interacts with the underlying Hardware for e.g. how Instagram interacts with a mobile phones camera



WHAT IS AN APPLICATION

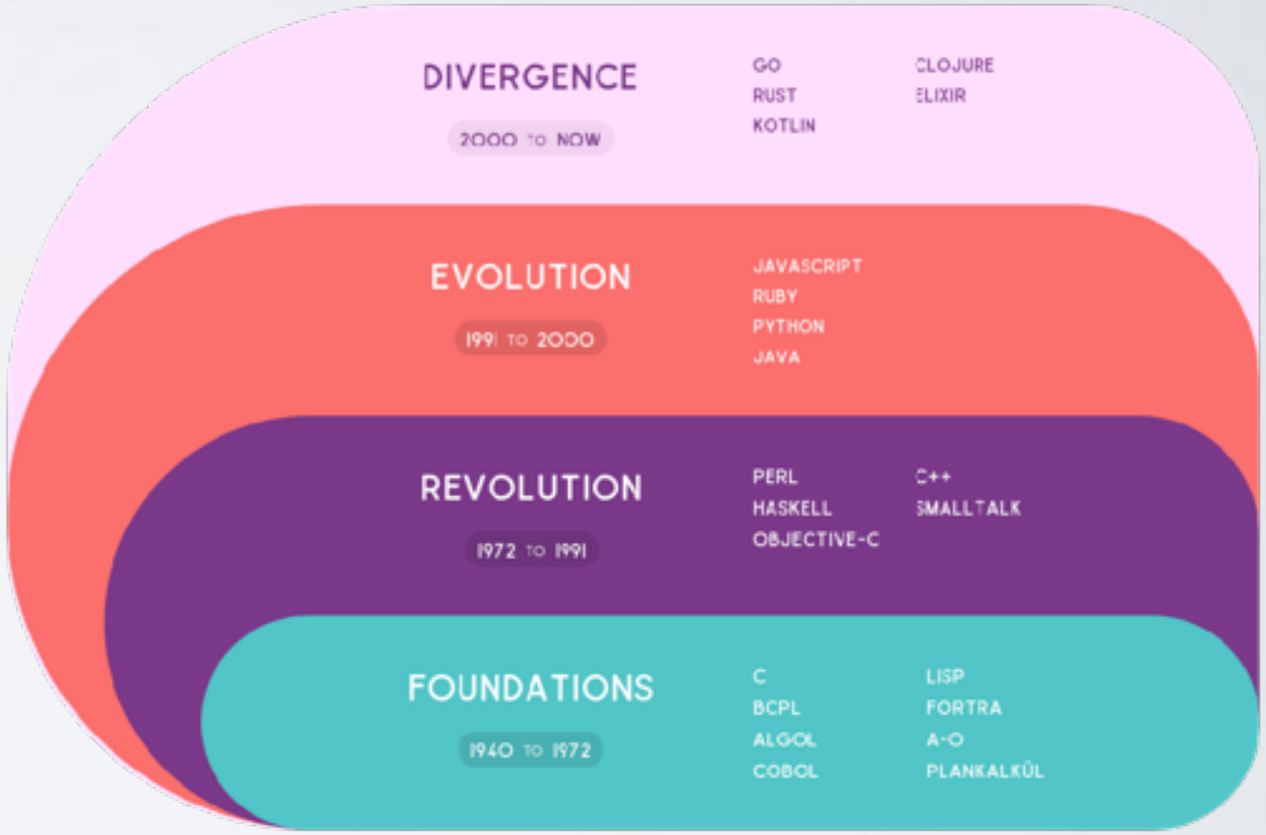
- Computers work by giving (**Programming**) them instructions for things you want them to do
- Different Applications give different instructions
- When an Application is started, a specific set of instructions are executed by the Computer
- The process of creating an Application is about producing the instructions (**Software**) that you want the Computer to execute

HOW TO CREATE SOFTWARE

- Computer *accept* instructions using 0's and 1's, Humans *create* instructions using words (speech or written)
- We need a way for Humans to *create* instructions (a **Language**) and for them to be translated (**Compiled**) into something Computers understand (the Application)

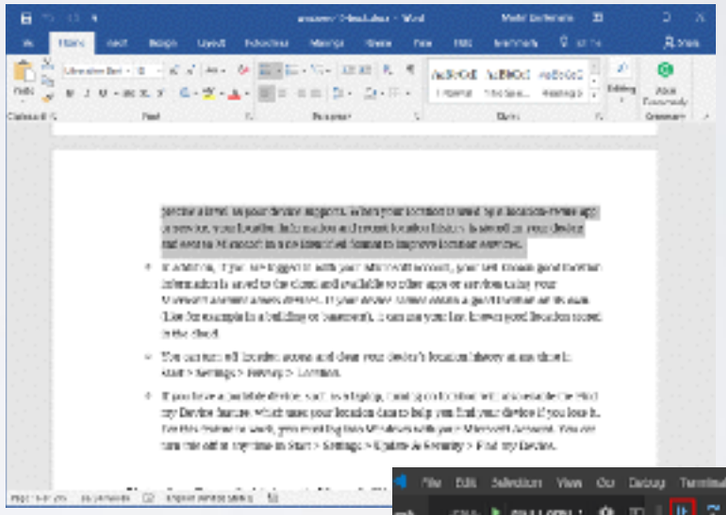
WHAT IS A PROGRAMMING LANGUAGE

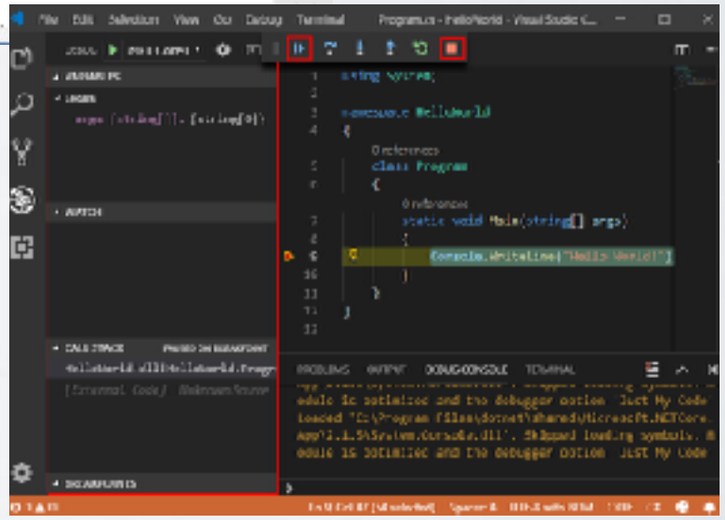
- A way for Humans to program instructions onto a Computer to tell it to do a specific task e.g. post a tweet
- There are a lot of different languages to choose from (see notes)
- Different Languages can be used to program the *same instructions*, in the same way that you can tell someone in English or French to 'close the door'



TOOLS TO MAKE IT EASIER

- Working with a Programming Language is like working with English
- We have tools to make it easier
- We use Microsoft Word when writing an Essay
- We use a **Integrated Development Environment** (IDE) when writing Application Software



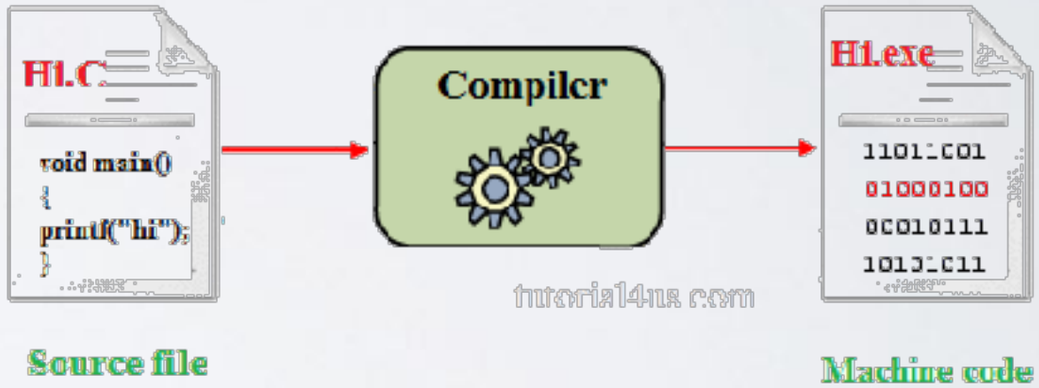


INTEGRATED DEVELOPMENT ENVIRONMENT

- Virtually every IDE will have a text editor designed to write and manipulate source code.
- Some tools may have visual components to drag and drop front-end components, but most have a simple interface with language-specific syntax highlighting.

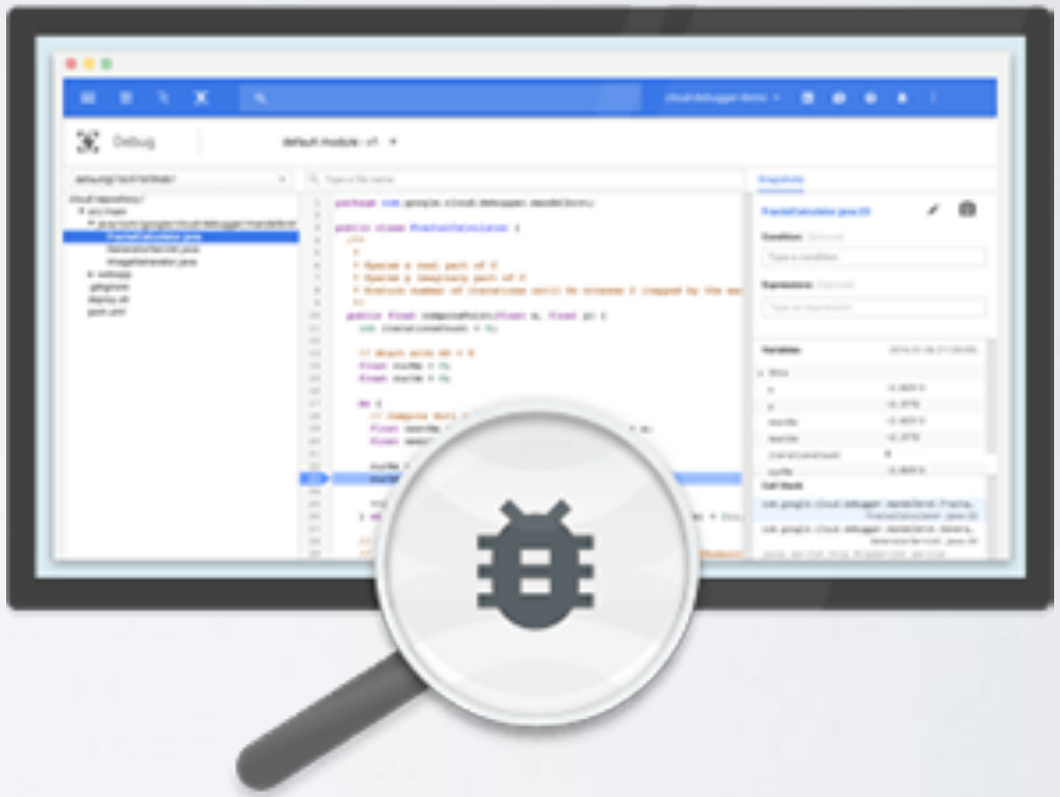
COMPILATION

- A **Compiler** is a application that translates instructions written in a Programming Language (source language) into another Programming Language (target language).
- The name Compiler is primarily used for programs that translate source code from a high-level programming language (C#) to a lower level language (machine code) to create an executable program (Application).



DEBUGGER

- Debugging tools assist users in identifying and remedying errors within source code.
- They often simulate real-world scenarios to test functionality and performance.
- Programmers and software engineers can usually test the various segments of code and identify errors before the application is released.



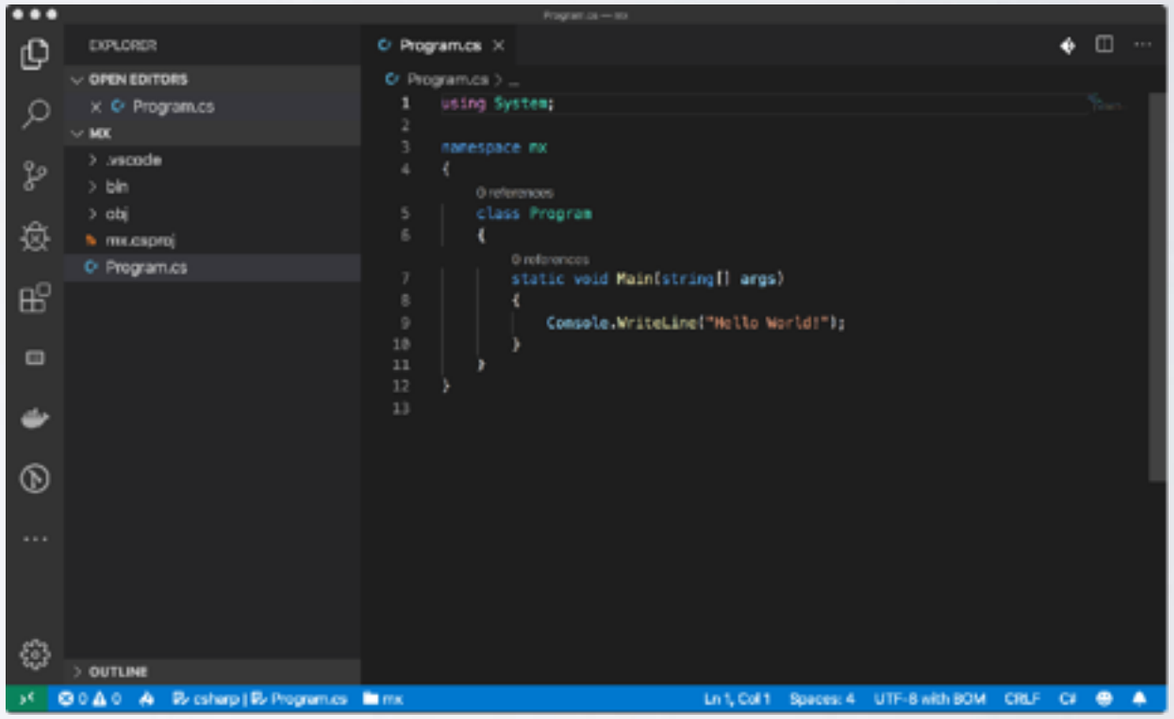
APPLICATION

GETTING STARTED - TOOLS

- IDE - https://code.visualstudio.com
- IDE Extension for C# - https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp
- C# Software Development Kit (SDK) - https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install

VSCODE

- Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS.
- It includes support for debugging, syntax highlighting, intelligent code completion, snippets, and code refactoring.
- Additional functionality can be added to it through extension plugins
- It is a simpler IDE for C# development than Visual Studio IDE



.NET SDK

- .NET Core Software Development Kit (SDK) is a set of libraries and tools that allow developers to create .NET Core applications and libraries. It contains the following components that are used to build and run applications:
  - The .NET Core CLI tools.
  - .NET Core libraries and runtime.
  - The dotnet driver.

Program.cs — mx

EXPLORER

OPEN EDITORS

Program.cs

MX

.vscode

bin

obj

mx.csproj

Program.cs

File List

Program.cs

Program.cs > ...

```
using System;

namespace mx
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

File Editor

PROBLEMS OUTPUT TERMINAL ... 1: bash

```
doneill at doneill-mbp in ~/Downloads/mx
$
```

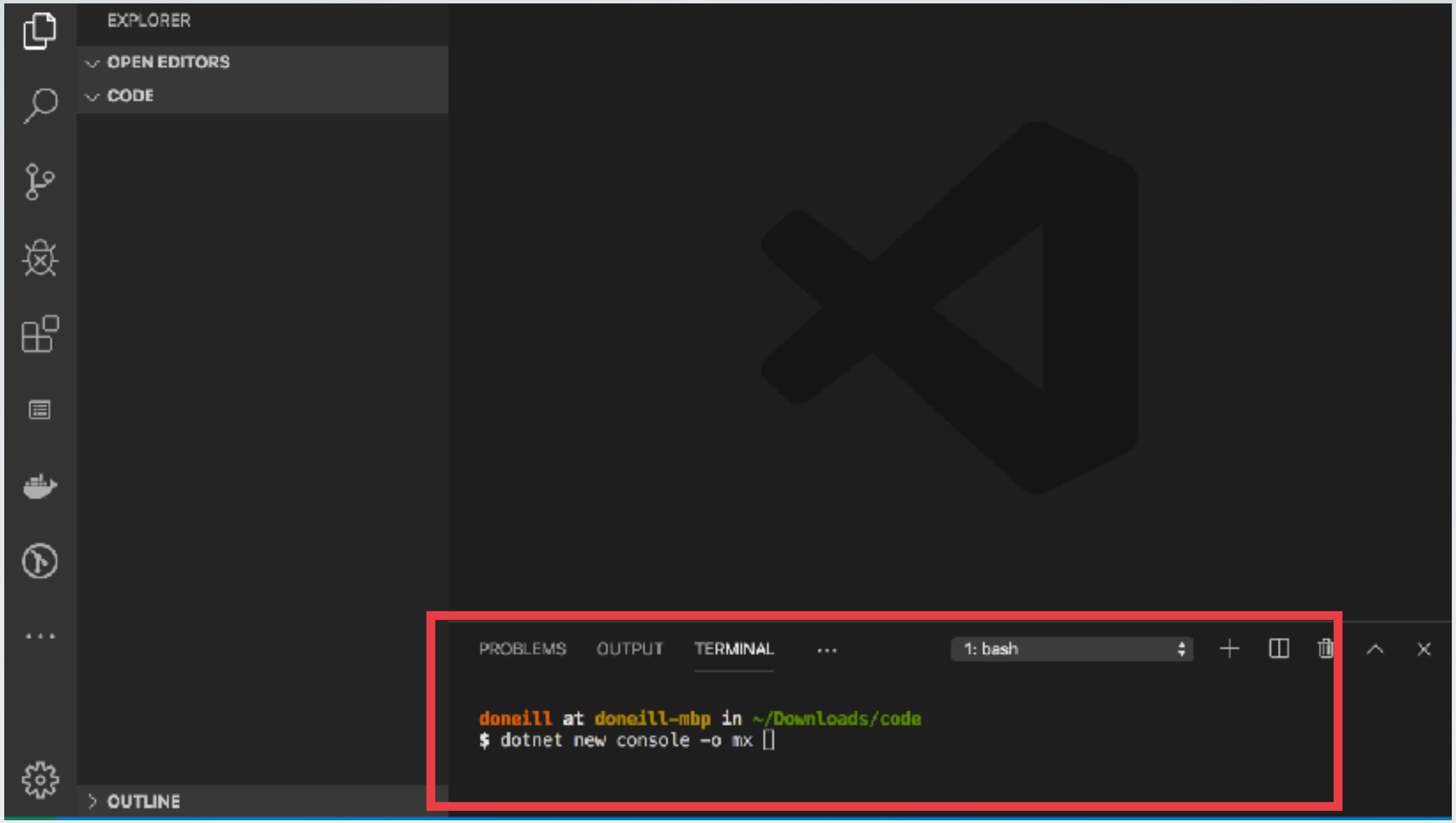
Terminal

OUTLINE

0 0 csharp | Program.cs mx Ln 1, Col 1 Spaces: 4 UTF-8 with BOM CRLF C#

HELLO WORLD

- Use the .NET SDK to create a skeleton cli application
- .NET SDK includes an CLI Application called 'dotnet', we will use this to 'scaffold' the files needed for the application.
- We will run the dotnet CLI command from within vscode using the terminal window



dotnet new console -o mx

HELLO WORLD

- This will create a variety of files in the Explorer window, including a file called Program.cs
- Program.cs contains the C# source code for our command line application
- The structure of this source file is described on the next screen

Program.cs — mx

Program.cs

Program.cs > ...

```
using System;

namespace mx
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Import types

Organise classes

Custom type

Application Entry Point

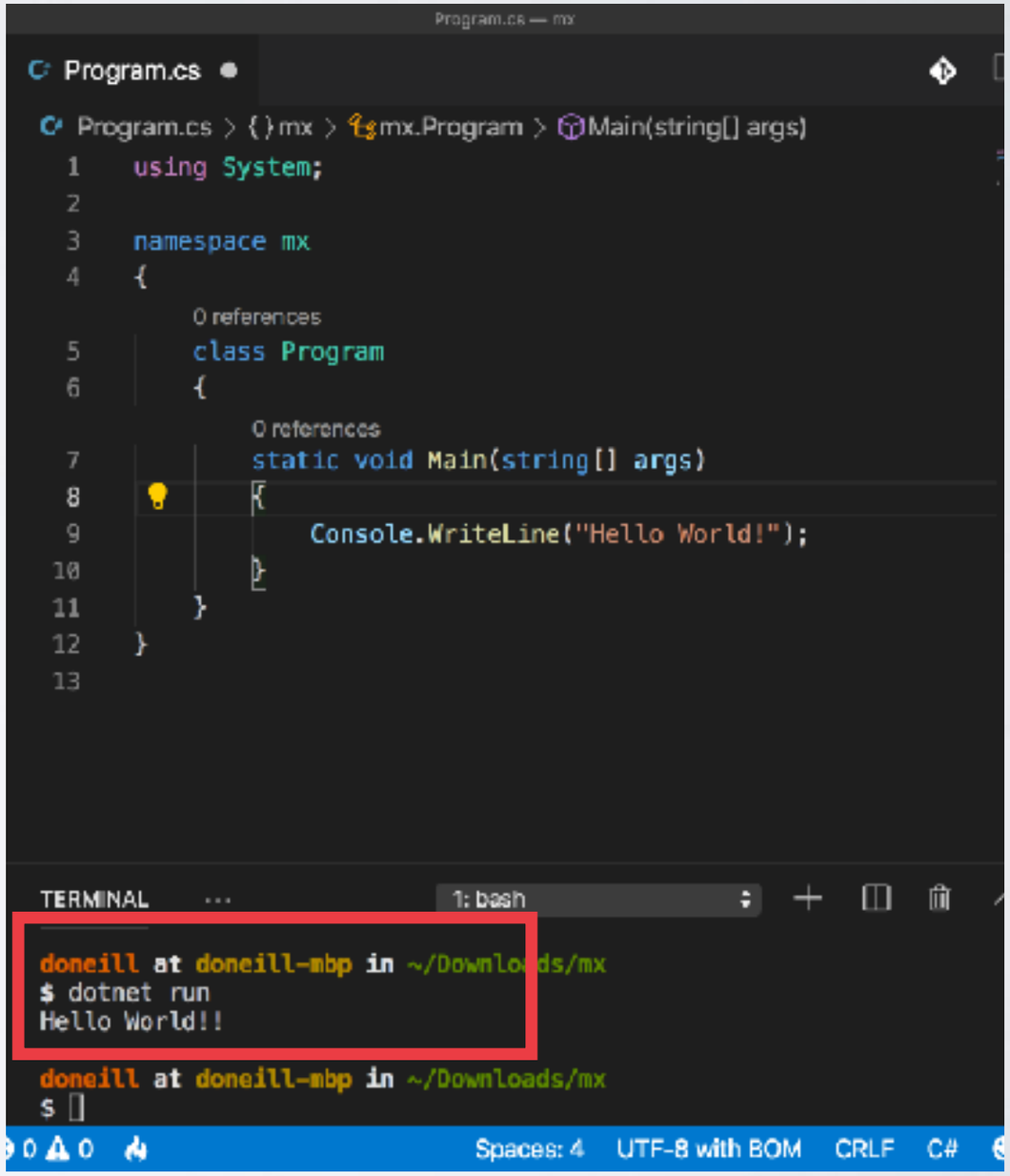
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
doneill at doneill-mbp in ~/Downloads/mx
$
```

0 0 csharp | Program.cs mx Ln 1, Col 1 Spaces: 4 UTF-8 with BOM CRLF C#

RUNNING THE APPLICATION

- We can run the application by typing **dotnet run** in the Terminal Window
- This will **Compile** (translate) the source code to a executable and run the application outputting the text 'Hello World!' to the terminal
- The application uses the **Class** Console and the **Method** WriteLine to output this text



DATA TYPES IN C#

- numeric
  - int (4 bytes), -2147483648 to 2147483647
- character
  - char (2 bytes), 0 to 128, examples; a,b,4
- string
  - string (20+ bytes), examples; "Hello World"
- boolean
  - bool (1 byte), false or true
- date/time
  - DateTime (8 bytes), 00:00:00 01/01/0001 to 23:59:59 31/12/9999, examples; 14:28:32 08/05/2019

Program.cs — mx

Program.cs

Program.cs > {}mx > mx.Program > Main(string[] args)

```
using System;

namespace mx
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            string message = "Hello World!!";

            Console.WriteLine(message);
        }
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
doneill at doneill-mbp in ~/Downloads/mx
$
```

0 0 csharp | Program.cs mx Ln 10, Col 1 Spaces: 4 UTF-8 with BOM CRLF C#

COLLECT SOME INPUT

- Now that we can write output to the Terminal, lets collect some input.
- If Console.WriteLine writes output to the Console (Terminal) then ...
- Console.Readline Reads input from the Terminal

Program.cs — mx

Program.cs | README.md

Program.cs > { } mx > mx.Program > Main(string[] args)

You, 6 minutes ago | 1 author (You)

```
1  using System;
2
```

You, 6 minutes ago | 1 author (You)

```
3  namespace mx
4  {
```

0 references | You, 6 minutes ago | 1 author (You)

```
5      class Program
6      {
```

0 references

```
7          static void Main(string[] args)
8          {
9              Console.Write("What is your name? ");
10             string input = Console.ReadLine();
11             string message = "Hello " + input + "!";
12             Console.WriteLine(message);
13         }
14     }
15 }
16
```

You, 6 minutes ago • Prompt for some text, collect

PROBLEMS 5 | OUTPUT | DEBUG CONSOLE | TERMINAL | 1: bash

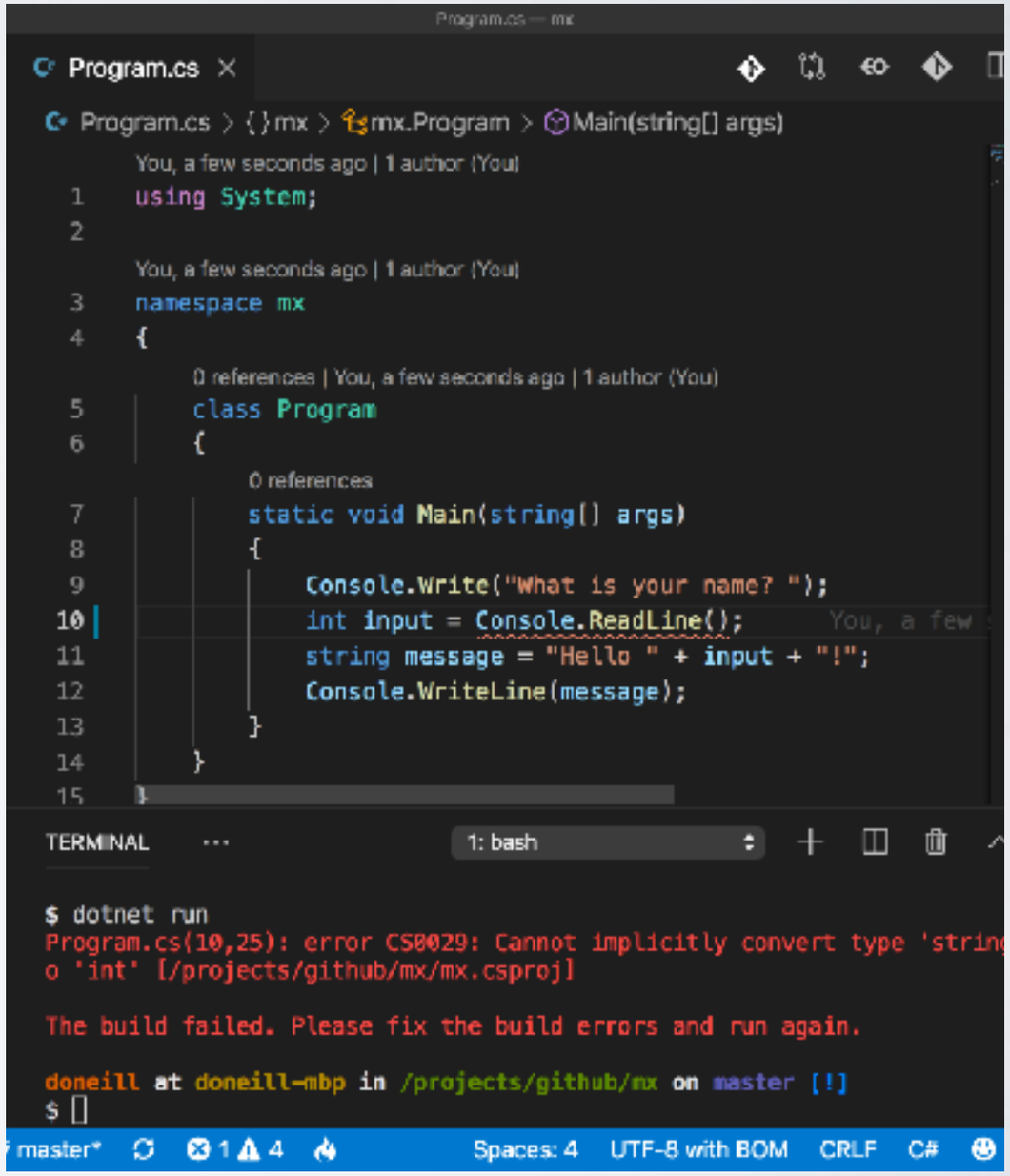
```
$ dotnet run
What is your name? Damian
Hello Damian!

doneill at doneill-mbp in /projects/github/mx on master [!]
$
```

master | 0 5 | csharp | Program.cs | Ln 9, Col 50 | Spaces: 4 | UTF-8 with BOM | CRLF | C# | 1

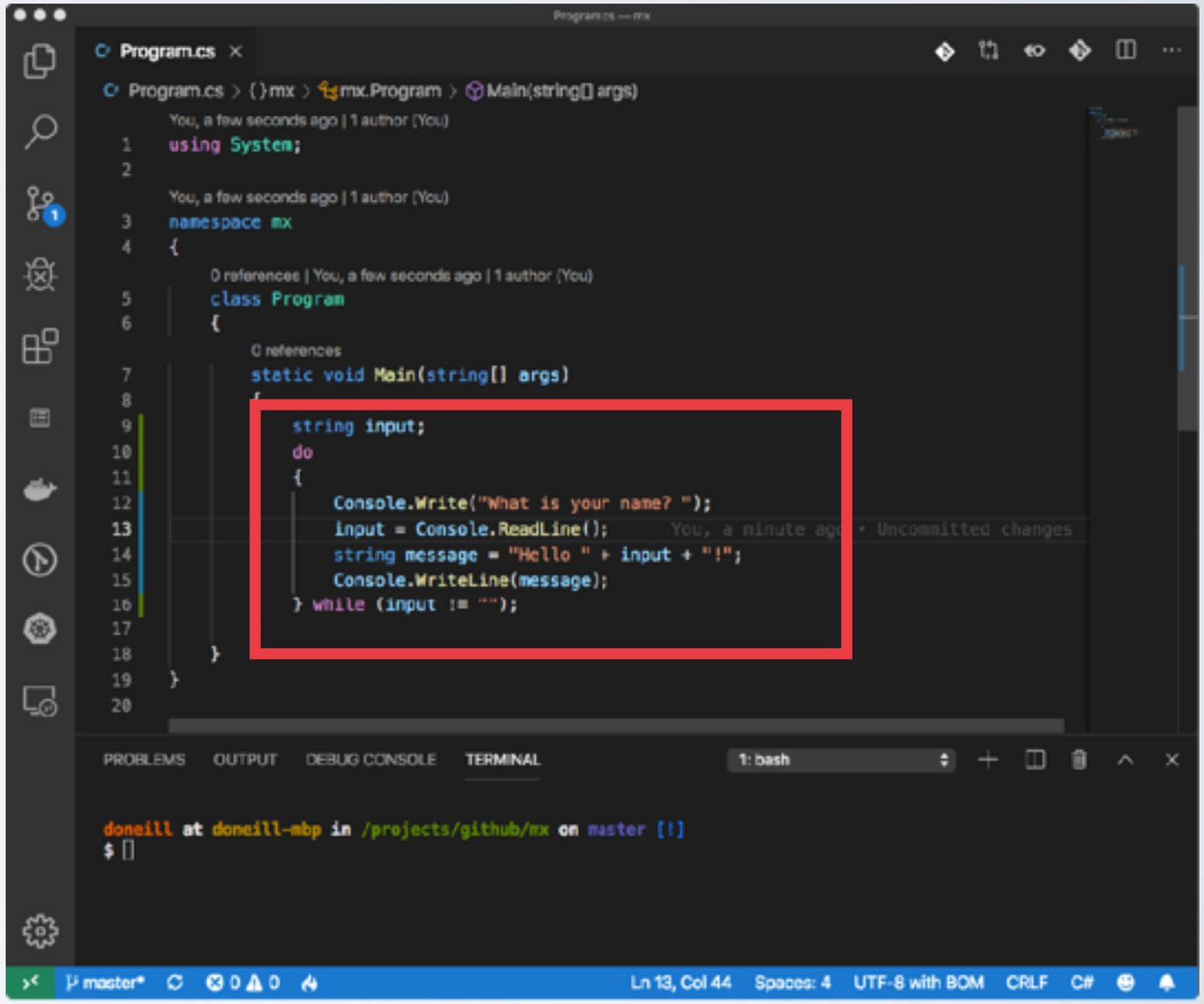
MIX AND MATCHING DATA TYPES

- What happens if we change the input type to be an int instead of string?
- **Syntax highlighting** in the IDE recognises there is a problem (red underline)
- Compiler complains that the method returns a string but try's to set the value to an int (not allowed)



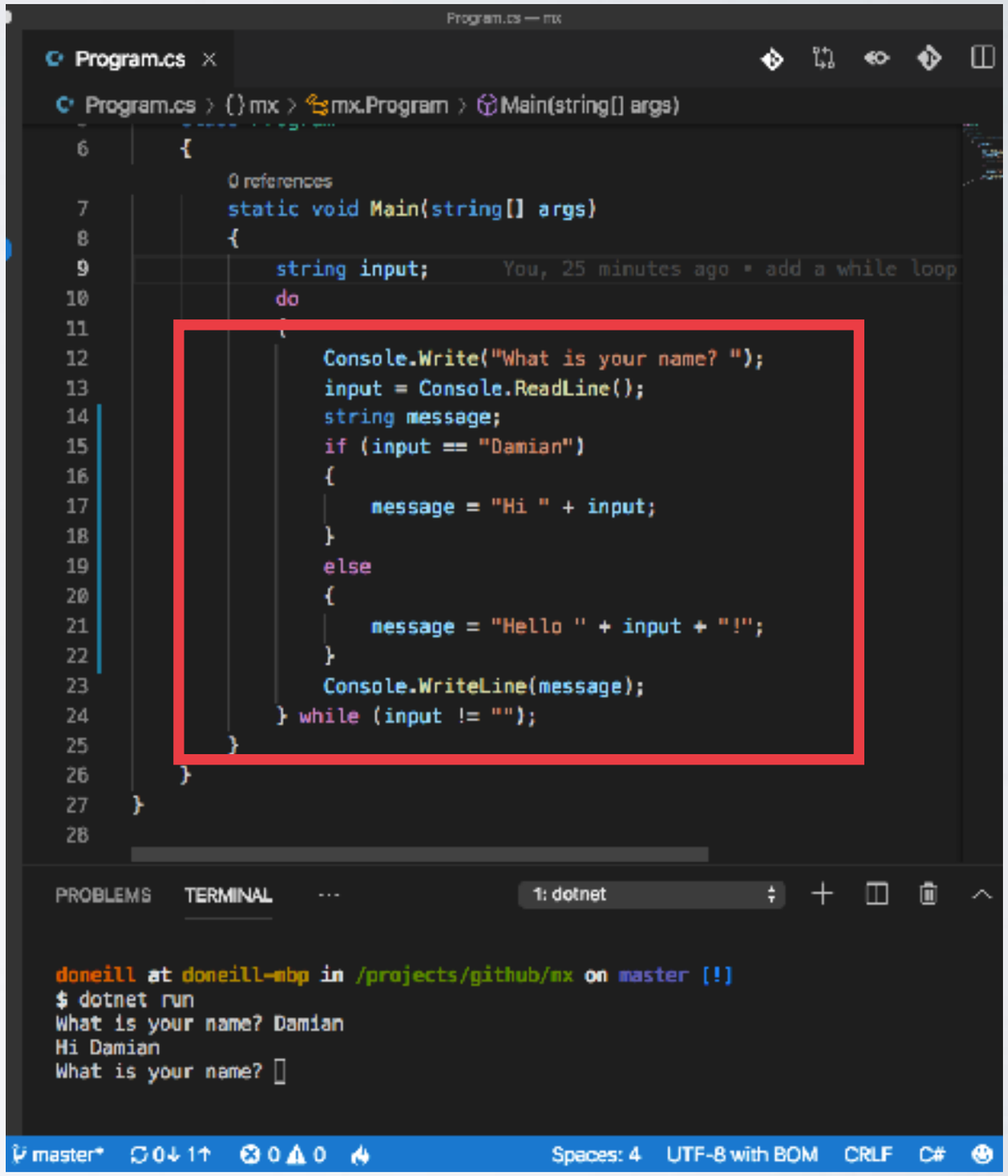
AGAIN AND AGAIN AND AGAIN

- Lots of times computers programs do the same thing over and over, this is called a **Loop**
- There are different ways of creating a loop in code
- One way is to **do** something **while** a condition is met (or not met)
- In the shown example the **condition** is that input is not empty



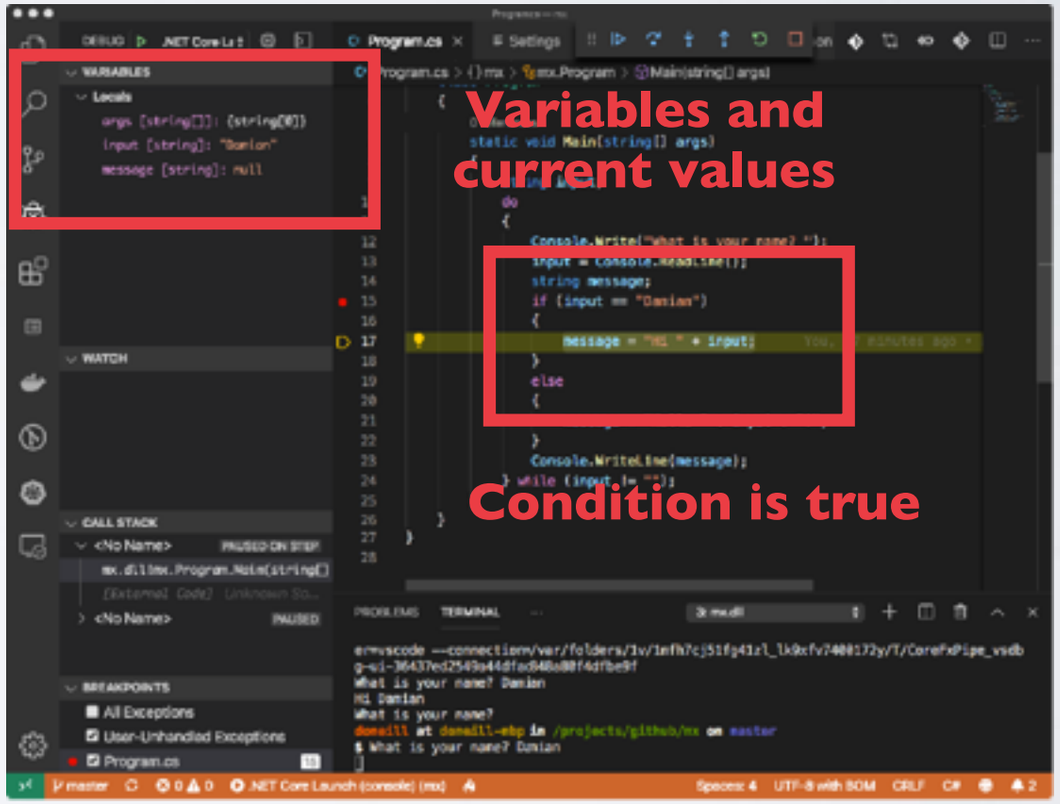
DECISION MAKING

- Software Applications make decisions all the time
- For e.g. in a Driving Application turn left or turn right, if the User a Child or Adult
- A common way of handling a decision is **if** a condition is valid (true) **then** do one thing **else** do another thing



DEBUGGER

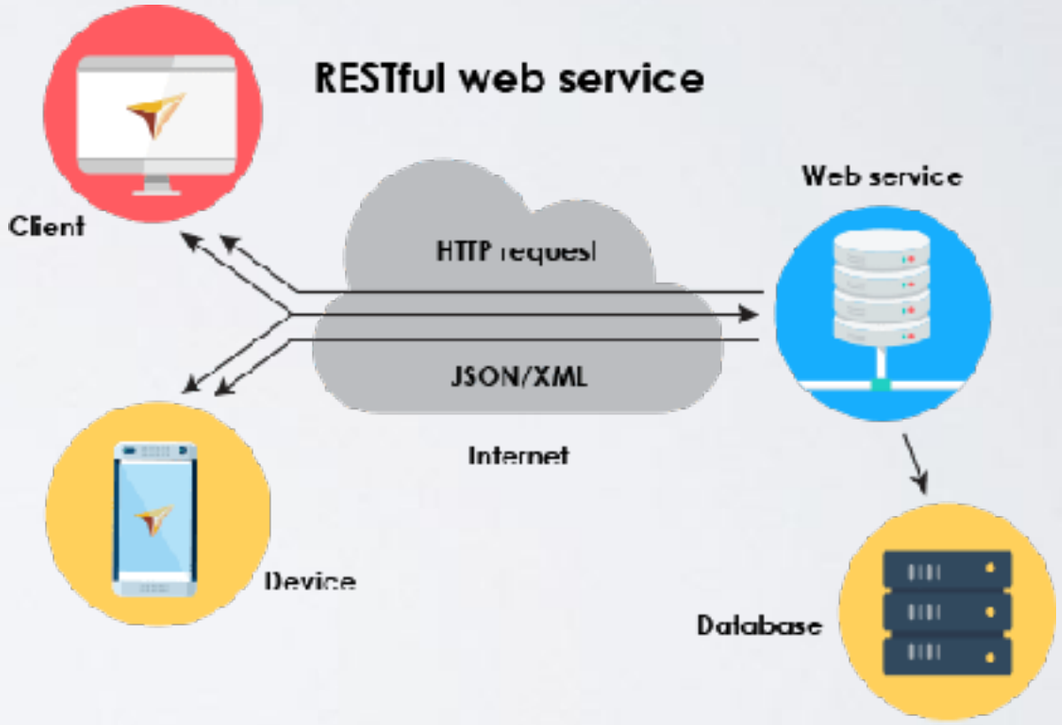
- When you debug your app, it usually means that you are running your application with the debugger attached.
- When you do this, the debugger provides many ways to see what your code is doing while it runs.
- You can step through your code and look at the values stored in variables, you can set watches on variables to see when values change, you can examine the execution path of your code, et al.



DAY TWO

SAMPLE APPLICATION

- The sample application will be command line based (no Graphical component)
- Query a Database, using a Web Service
- Build a set of features around the returned data



WEB APIS

- Application Programming Interface (API) is some kind of interface which has a set of functions that allow programmers to access specific features or data of an application, operating system or other services.
- Web API as the name suggests, is an API over the web which can be accessed using HTTP protocol.
- https://github.com/public-apis/public-apis - list of public apis that can be used
- https://promotocrossapi.com/docs - **sample web api** that we will use for this application

Pro Motocross API

promotocrossapi.com

Home Documentation Contact

Pro Motocross API

Access the Lucas Oil AMA Pro Motocross & Monster Energy Supercross series data

Get Started →

#1 Pro Motocross & Supercross API

We've made it easier than ever to develop your moto apps using flexible search options with rapid response times. This makes it simple to build exciting data driven visualizations.

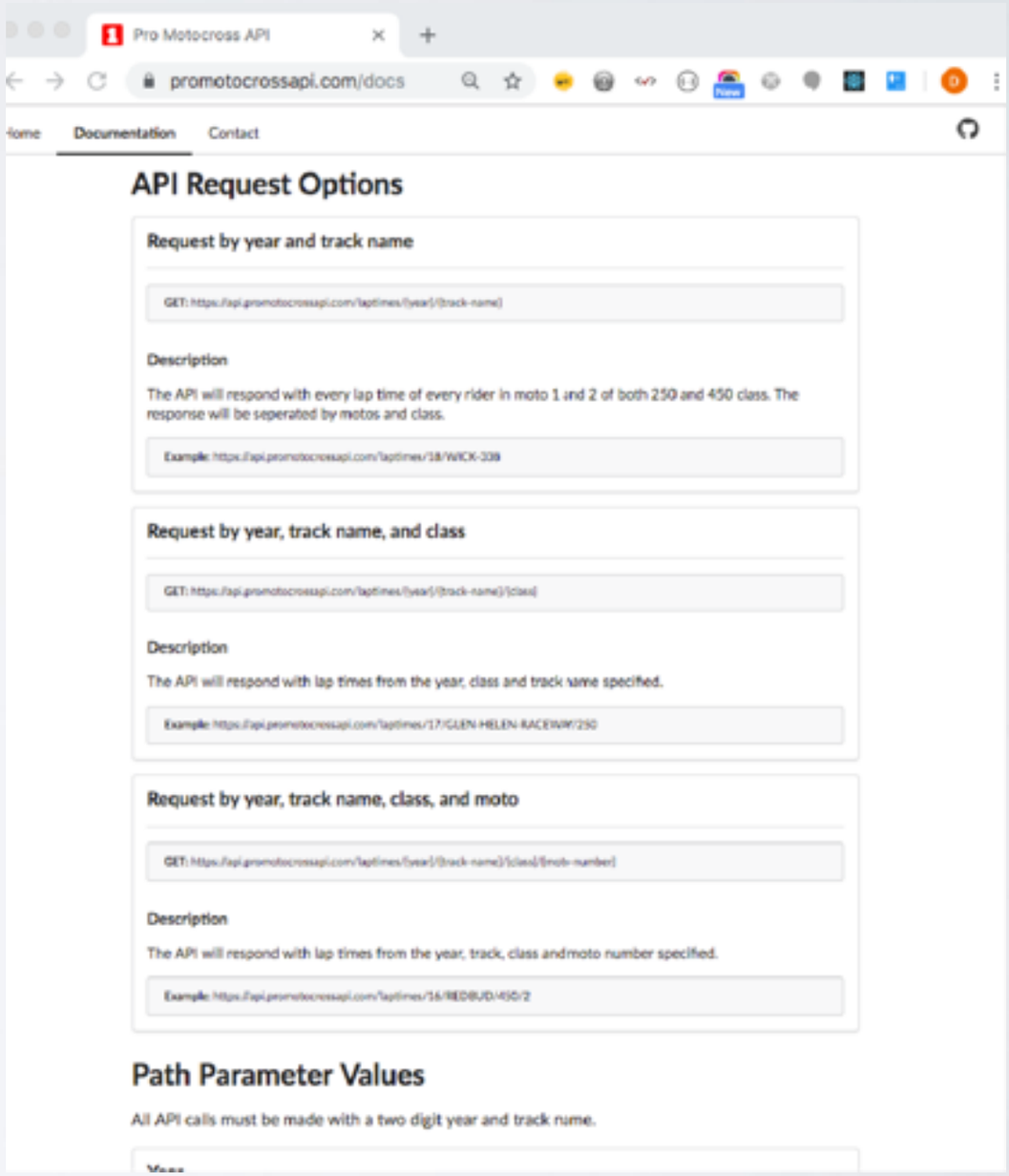
Help Build The Moto Community

Building a moto app? We'd love to hear about it and help get the word out.

Contact

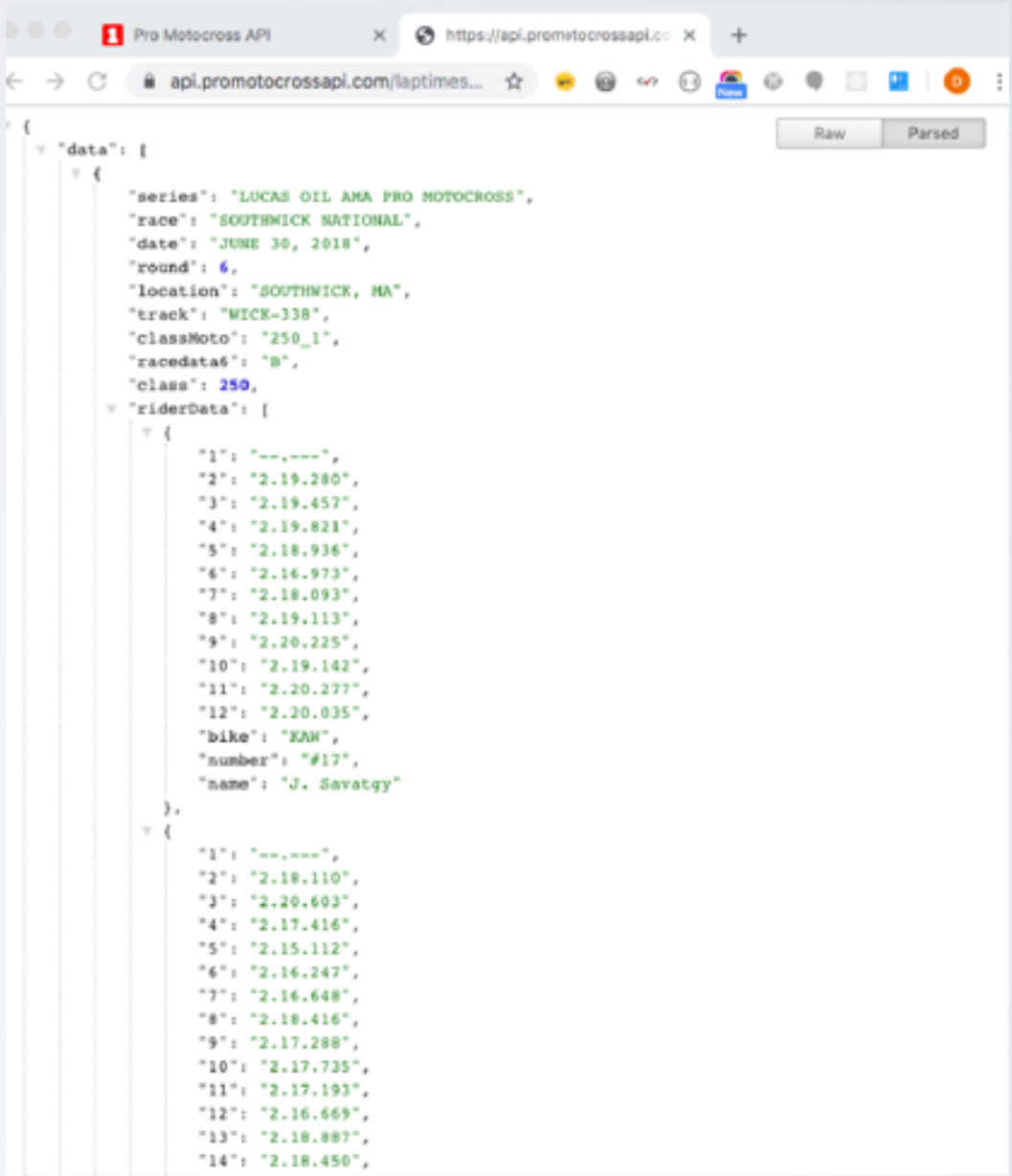
SAMPLE WEB SERVICE

- Pro Motocross website includes documents on **what** Requests can be made and **how**
- For example calling https://api.promotocrossapi.com/laptimes/18/WICK-338 will return a dataset from 20**18** for all the races at the **WICK-338** track.



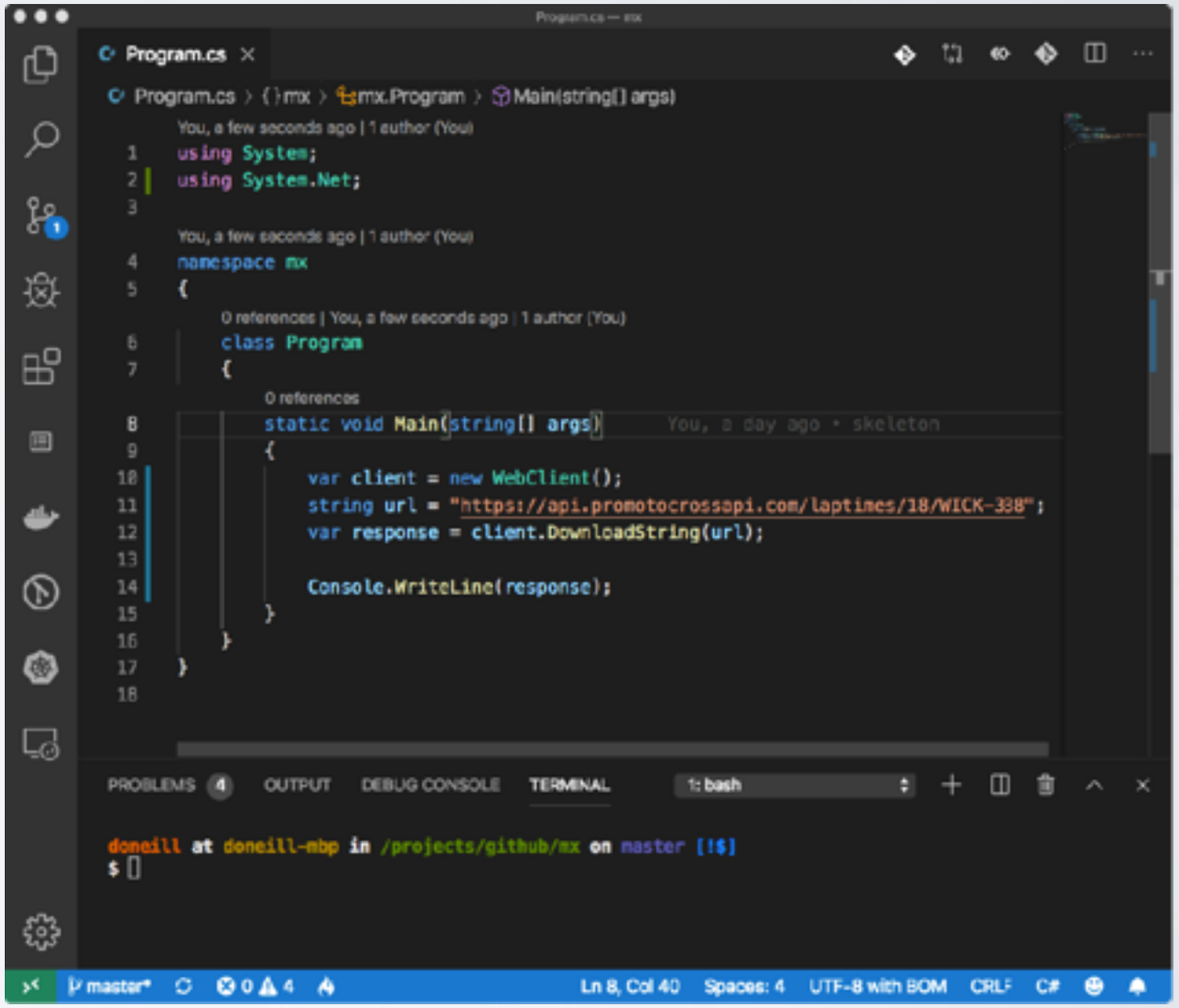
DATABASE QUERY RESULTS

- Click https://api.promotocrossapi.com/laptimes/18/WICK-338 in browser
- Screenshot shows a subset of the data that was returned
- Returned information is in JSON format
- We will need a **library** later to make working with JSON easier



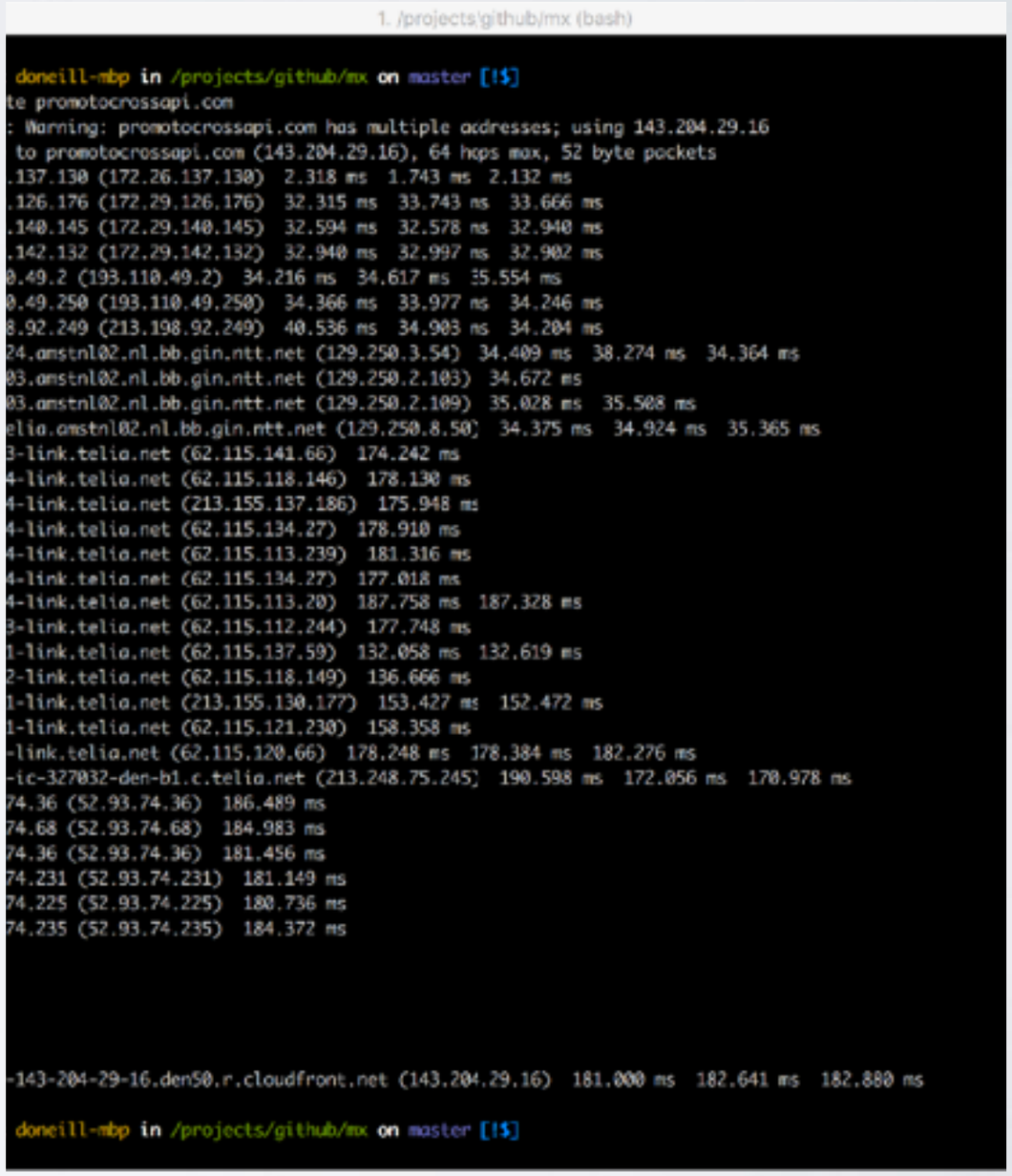
SAMPLE DATABASE REQUEST

- Use a Class (WebClient) from **Library** system.Net to request the data
- Store the data in a **string type** called response
- Write the value of response to the Console



WHERE IS THE DATA COMING FROM

- The website is hosted at 143.204.29.16
- We can find where this is at by using a geo lookup http://geoiplookup.net/ip/143.204.29.16
- This Database is running in Seattle, USA
- We can see the route to request went to get there by running a traceroute

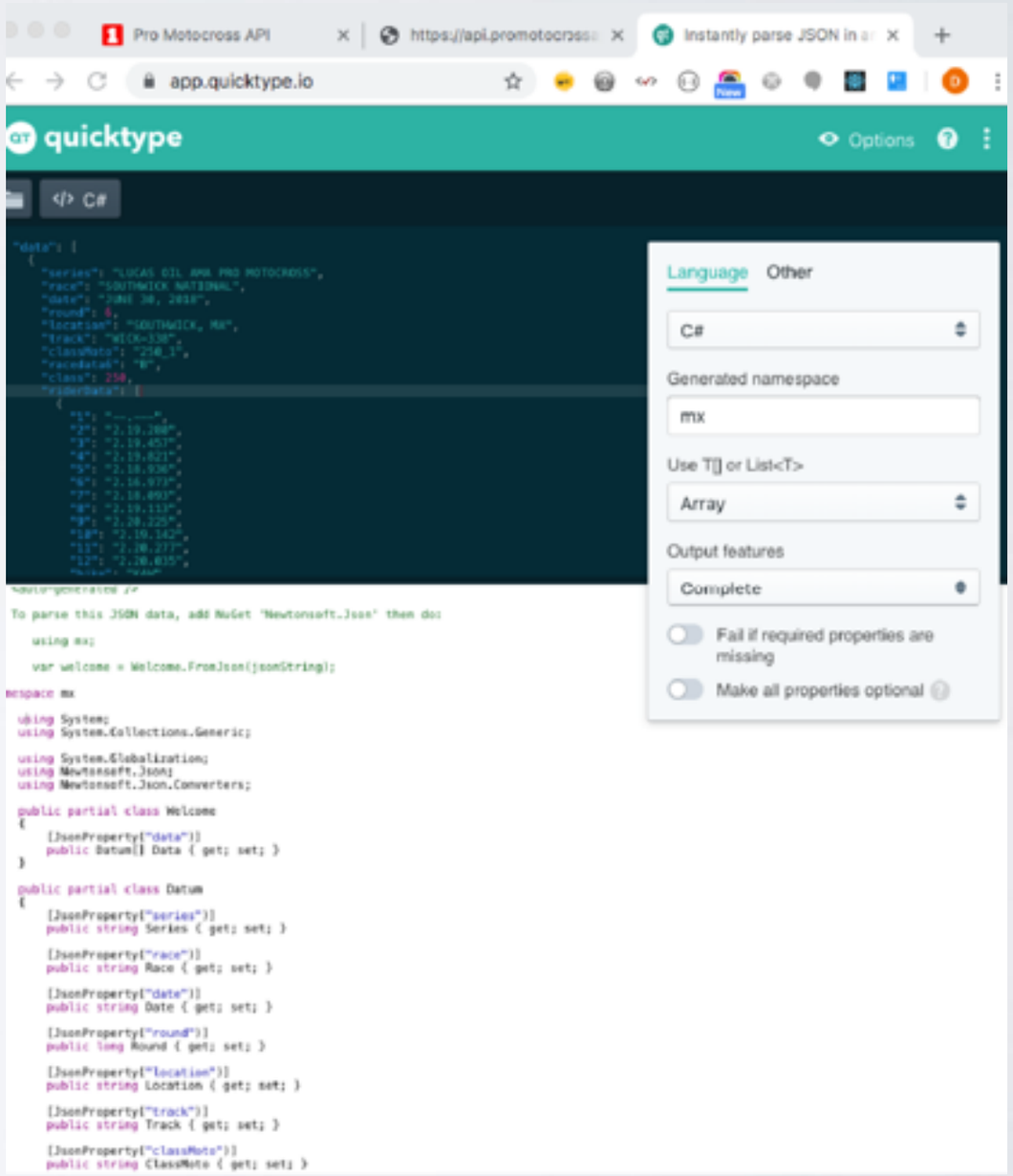


HANDLE JSON DATA

- We can use a tool to make it easy to work with the JSON data in C#
- https://app.quicktype.io - insert JSON and the web page will create an autogenerated type for working with the JSON

GENERATE DATA TYPE

- Copy the JSON data into the box on the left
- Select the language C#
- Name the Namespace **mx**
- Code is generated on the bottom of the screen
- Save this in a file named Welcome.cs in the same directory as Program.cs



Welcome.cs — mx

EXPLORER

OPEN EDITORS

MX

- .vscode
- bin
- docs
- obj
- .gitignore
- mx.csproj M
- Program.cs
- README.md 4
- Welcome.cs U

Program.cs | Welcome.cs

Welcome.cs > {}mx

```
// <auto-generated />
//
// To parse this JSON data, add NuGet 'Newtonsoft.Json
//
//    using mx;
//
//    var welcome = Welcome.FromJson(jsonString);

namespace mx
{
    using System;
    using System.Collections.Generic;

    using System.Globalization;
    using Newtonsoft.Json;
    using Newtonsoft.Json.Converters;

    3 references
    public partial class Welcome
    {
        [JsonProperty("data")]
        0 references
        public Datum[] Data { get; set; }
    }
```

OUTPUT .NET

```
  Restore completed in 195.18 ms for /projects/github/mx/
  mx.csproj.
Done: 0.
```

OUTLINE

master* 0 4 6 csharp | Welcome.cs Spaces: 4 UTF-8 LF C#

USING THE NEW TYPE

- The mainline can now be updated to handle the JSON data returned from the Database using the autogenerated **data type**
- This will make using the data much easier

Program.cs — mx

Program.cs

Program.cs > { } mx > mx.Program > Main(string[] args)

You, 4 minutes ago | 1 author (You)

```
4   namespace mx
5   {
        0 references | You, 4 minutes ago | 1 author (You)
6       class Program
7       {
            0 references
8           static void Main(string[] args)
9           {
10              var client = new WebClient();
11              string url = "https://api.promotocrossapi.com/laptimes/18/WICK-338";
12              var response = client.DownloadString(url);
13
14              var mx = Welcome.FromJson(response);
15
16              Console.WriteLine("There was " + mx.Data.Length + " Races in 2018 at the WICK-338 track");
17          }
18      }
19  }
20
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
doneill at doneill-mbp in /projects/github/mx on master [!$]
$ dotnet run
There was 4 Races in 2018 at the WICK-338 track

doneill at doneill-mbp in /projects/github/mx on master [!$]
$
```

master* 0 4 csharp | Program.cs mx You, a few seconds ago Ln 16, Col 61 Spaces: 4 UTF-8 with BOM CRLF C#

MAKING A SECOND REQUEST

- We can now call the Web Request again and change the query we send to the database
- We can choose a different track and different year
- The existing query will run, then the second query

Program.cs — mx

Program.cs

Program.cs > {}mx > mx.Program > Main(string[] args)

```
0 references
8   static void Main(string[] args)
9   {
10      var client = new WebClient();
11
12      string url = "https://api.promotocrossapi.com/laptimes/18/WICK-338";
13      var response = client.DownloadString(url);
14
15      var mx = Welcome.FromJson(response);
16
17      Console.WriteLine("There was " + mx.Data.Length + " Races in 2018 at the WICK-338 track");
18
19      url = "https://api.promotocrossapi.com/laptimes/17/HANGTOWN";      You, a few seconds ago • Uncommitted
20      response = client.DownloadString(url);
21      mx = Welcome.FromJson(response);
22
23      Console.WriteLine("There was " + mx.Data.Length + " Races in 2017 at the HANGTOWN track");
24  }
25  }
26 }
27
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
doneill at doneill-mbp in /projects/github/mx on master [!$]
$ dotnet run
There was 4 Races in 2018 at the WICK-338 track
There was 4 Races in 2017 at the HANGTOWN track

doneill at doneill-mbp in /projects/github/mx on master [!$]
$
```

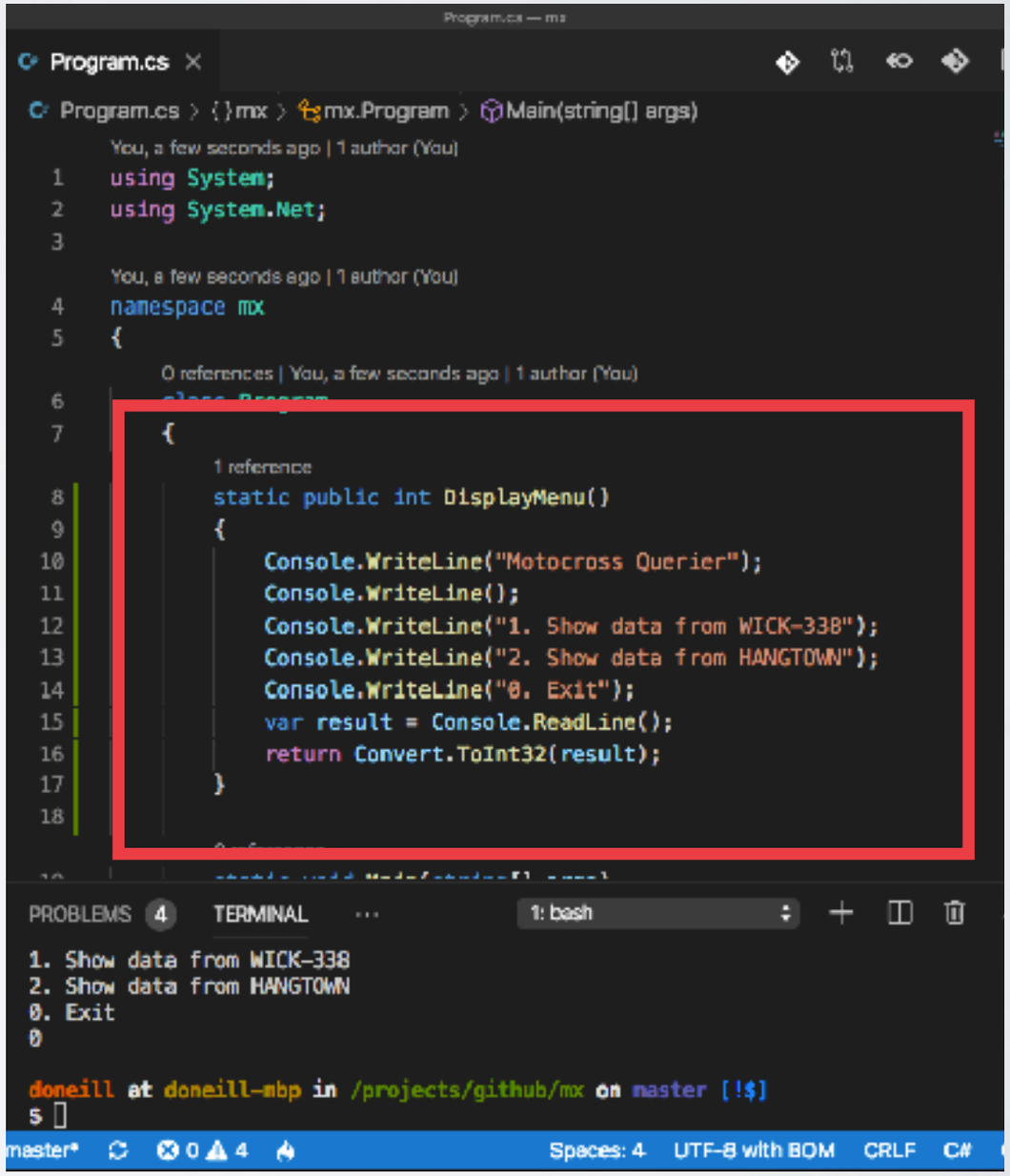
master* 0 4 csharp | Program.cs mx You, a few seconds ago Ln 19, Col 63 Spaces: 4 UTF-8 with BOM CRLF C#

LET THE USER CHOOSE

- We'll introduce a Menu **Function** where the user can choose which request to make
- A Console Menu is like a food Menu, you get a set of choices and you choose the one you want

MENU FUNCTION

- Create a **User Defined Function** for displaying a Menu of options
- The User Function writes a Menu to the Console
- And reads a choice from the Console



UPDATE THE MAINLINE TO USE THE MENU

- Create a **Loop** that calls the **User function** to Display the Menu
- In the Loop, based on the User Input, make a **decision** about what Request to send
- Exit the Loop when the **condition** is met

Program.cs — mx

Program.cs

Program.cs > {} mx > mx.Program > Main(string[] args)

```
18
        0 references
19      static void Main(string[] args)
20      {
21          var client = new WebClient();
22          int userInput = 0;
23          string url = "", track = "", response;
24          Welcome mx;
25          do
26          {       You, a few seconds ago • Uncommitted changes
27              userInput = DisplayMenu();
28              if (userInput == 1)
29              {
30                  url = "https://api.promotocrossapi.com/laptimes/18/WICK-338";
31                  track = "WICK-338";
32              }
33              if (userInput == 2)
34              {
35                  url = "https://api.promotocrossapi.com/laptimes/18/HANGTOWN";
36                  track = "HANGTOWN";
37              }
38              if (userInput != 0)
39              {
40                  response = client.DownloadString(url);
41                  mx = Welcome.FromJson(response);
42                  Console.WriteLine("There was " + mx.Data.Length + " Races in 2018 at the " + track + " track");
43              }
44          } while (userInput != 0);
45      }
46  }
47  }
48
```

Loop

Decision

Loop Exit Condition

master* | 0 4 | csharp | Program.cs | mx | You, a few seconds ago | Ln 26, Col 14 | Spaces: 4 | UTF-8 with BOM | CRLF | C#

DAY THREE

FIXED LOOPS

- Some Loops run a fixed number of times, for e.g. a Loop that runs 7 times to simulate the days in a week

- For these types of loops we use the for keyword

```
for (variable initialisation; condition; steps)
 {
    //execute this code block as long as condition is satisfied
 }
```

Program.cs — mx

Program.cs

Program.cs > {}mx > mx.Program > Main(string[] args)

```
25      do
26      {
27          userInput = DisplayMenu();
28          if (userInput == 1)
29          {
30              url = "https://api.promotocrossapi.com/laptimes/18/WICK-338";
31              track = "WICK-338";
32          }
33          if (userInput == 2)
34          {
35              url = "https://api.promotocrossapi.com/laptimes/18/HANGTOWN";
36              track = "HANGTOWN";
37          }
38          if (userInput != 0)
39          {
40              response = client.DownloadString(url);
41              mx = Welcome.FromJson(response);
42              Console.WriteLine("There was " + mx.Data.Length + " Races in 2018 at the " + track + " track");
43              for (int i = 0; i < mx.Data.Length; i++)
44              {
45                  Console.WriteLine("Round " + mx.Data[i].Round + " Race " + i + ": " + mx.Data[i].Class + " Moto " + mx.Data[i].Moto);
46              }
47          }
48      } while (userInput != 0);
49
50
51
52
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL 3: mx.dll

```
1
There was 4 Races in 2018 at the WICK-338 track
Round 6 Race 0: 250 Moto 1
Round 6 Race 1: 250 Moto 2
Round 6 Race 2: 450 Moto 1
```

master* 0 0 2 .NET Core Launch (console) (mx) csharp | Program.cs Ln 45, Col 140 Spaces: 4 UTF-8 with BOM CRLF C# 2

LOOPS WITHIN LOOPS

Program.cs — mx

Program.cs > {} mx > mx.Program > Main(string[] args)

```
            if (userInput != 0)
            {
                response = client.DownloadString(url);
                mx = Welcome.FromJson(response);
                Console.WriteLine("There was " + mx.Data.Length + " Races in 2018 at the " + track + " track");
                for (int i = 0; i < mx.Data.Length; i++)
                {
                    Console.WriteLine("Round " + mx.Data[i].Round + " Race " + i + ": " + mx.Data[i].Class + " Moto " + mx.Data[i].Moto);
                    for (int j = 0; j < mx.Data[i].RiderData.Length; j++)
                    {
                        Console.WriteLine(mx.Data[i].RiderData[j].Name + " rode a " + mx.Data[i].RiderData[j].Bike);
                    }
                }
            }
        } while (userInput != 0);
    }      You, 2 days ago • skeleton
    }
}
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL 3: mx.dll

```
B. Lamay rode a Hon

Motocross Querier

1. Show data from WICK-338
2. Show data from HANGTOWN
0. Exit
```

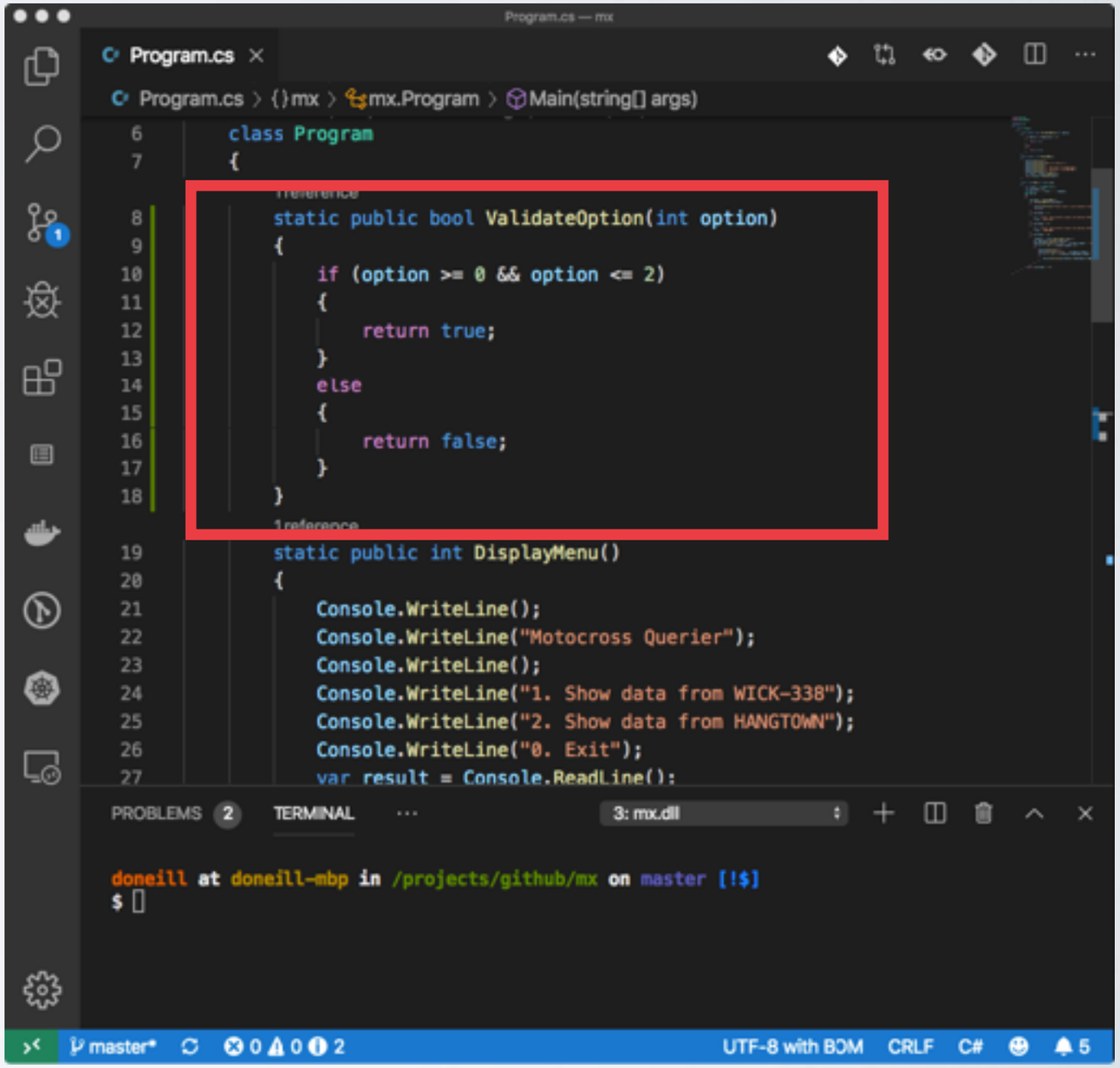
master* 0↓ 1↑ 0 0 2 .NET Core Launch (console) (mx) csharp | Program.cs mx You, 2 days ago Ln 55, Col 10 Spaces: 4 UTF-8 with BOM CRLF C# 2

VALIDATION (**PLTFR**)

- **P**resence - e.g. Not Blank, File exists, etc.
- **L**ength - e.g. 10 Digits Long
- **T**ype - e.g. is a Number
- **F**ormat - e.g. Specific Format (dd/mm/yyyy)
- **R**ange - e.g. a value between 10-20

VALIDATE INPUT

- Lets add some code to validate our Menu's input
- Create a User Defined Function for checking the value input by the User
- Implement a Range Check with a **Logical Operator** AND (&&)
- Use the **Relational Operators** Greater Than or Equal To (>=) and Less Than or Equal To (<=)



CHECK INPUT

- Now update our loop to check the input every time the User make a choice
- If the check fails, output a warning to the User
- Exit this iteration of the Loop since the input is valid

```
Program.cs
Program.cs > {}mx > mx.Program > Main(string[] args)
31          static void Main(string[] args)
32          {
33              var client = new WebClient();
34              int userInput = 0;
35              string url = "", track = "", response;
36              Welcome mx;
37              do
38              {
39                  userInput = DisplayMenu();
40                  if (!ValidateOption(userInput))
41                  {       You, a few seconds ago • Uncommitted changes
42                      Console.WriteLine("Please input a value between 0 and 2");
43                      continue;
44                  }
45                  if (userInput == 1)
46                  {
47                      url = "https://api.promotocrossapi.com/laptimes/18/WICK-338";
48                      track = "WICK-338";
49                  }
50                  if (userInput == 2)
51                  {
52                      url = "https://api.promotocrossapi.com/laptimes/18/HANGTOWN";
53                      track = "HANGTOWN";
54                  }
PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL   3: mx.dll

doneill at doneill-mbp in /projects/github/mx on master [!$]
$
master*   0 0 2   .NET Core Launch (console) (mx)   Spaces: 4   UTF-8 with BOM   CRLF   C#
```

WRITING TO FILE

- Check if a file (at a specific **path**) exists, if not notify the Users - **Presence Check** (Validation)
- Open a File for **Appending**
- Write a Line of text to the file including a **DateTime** data type

Program.cs — mx

Program.cs

Program.cs > {}mx > mx.Program > Main(string[] args)

```
67                      Console.WriteLine(mx.Data[i].RiderData[j].Name + " rode a " + mx.
68                  }
69              }
70
71              String path = @"datafile.txt";
72              if (!File.Exists(path)) // Presence Check
73              {
74                  Console.WriteLine("No such file existed; one was created.");
75              }
76              using (StreamWriter sw = File.AppendText(path))
77              {
78                  sw.WriteLine(mx.Data[0].Track + " selected at " + DateTime.Now);
79              }
80          You, a few seconds ago • Uncommitted changes
81              }
82          } while (userInput != 0);
83      }
84  }
85  }
86
```

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
doneill at doneill-mbp in /projects/github/mx on master [!$]
$
```

master* 0 tests 0 4 2 Watch Ln 80, Col 1 Spaces: 4 UTF-8 with BOM CRLF C#

THANKS