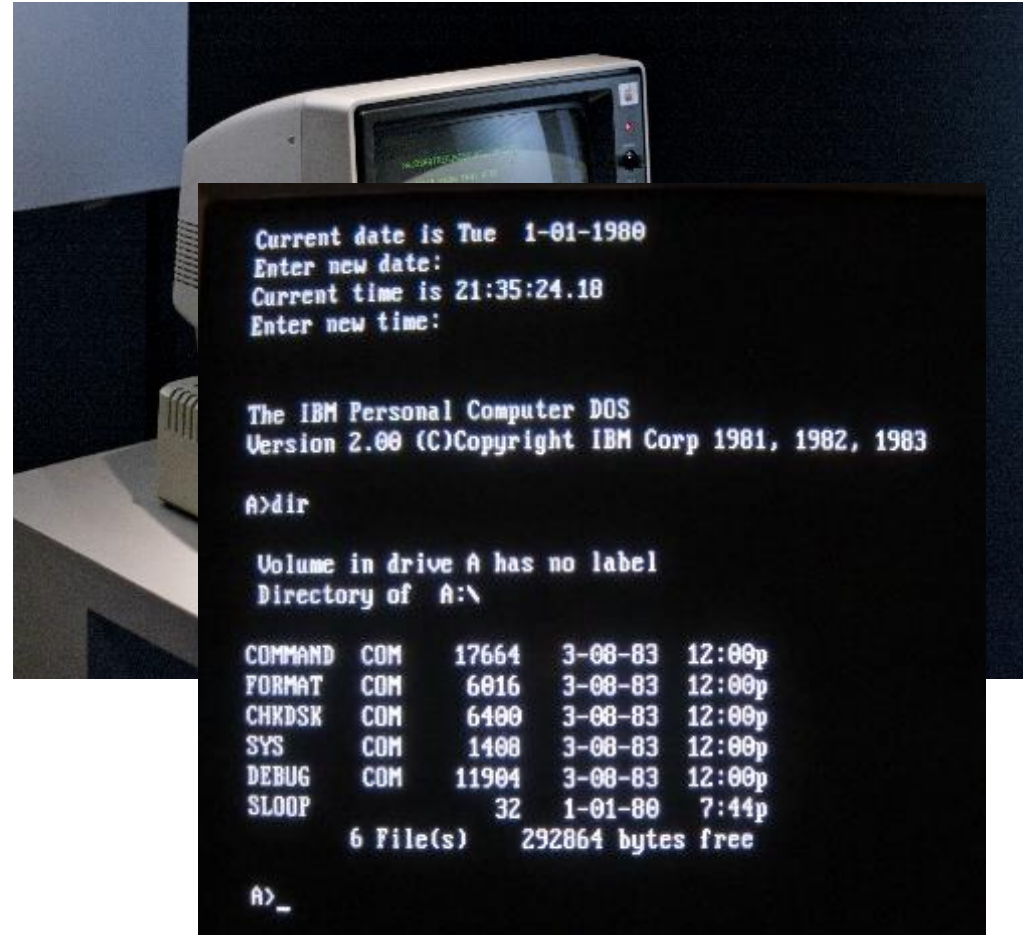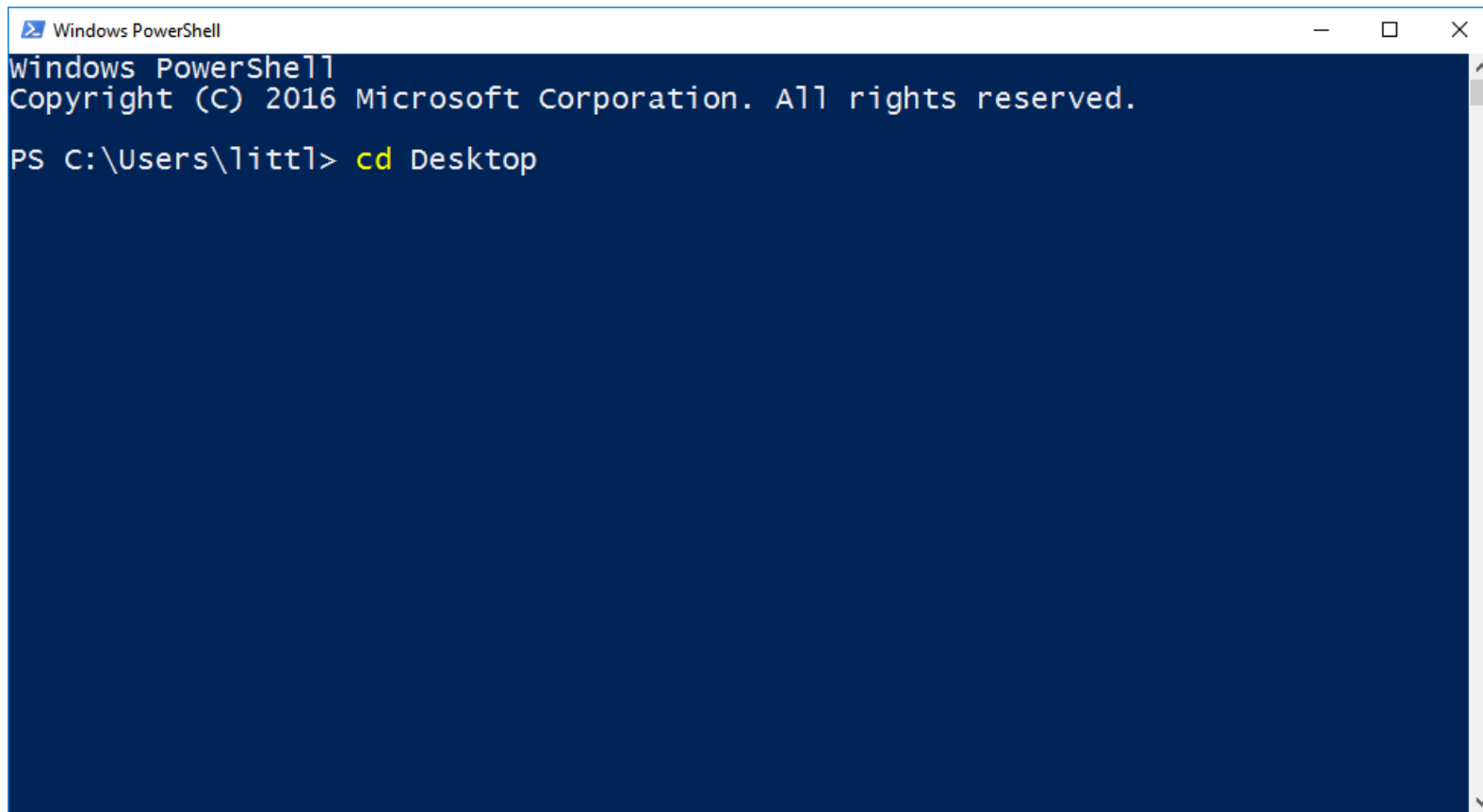# Hello

LAB 2

BME 121 2016

Jeff Luo

Topics

- PowerShell
- Managing BME 121 Code Projects
- Basic Structure of a Program
- Comments, Variables, Console Input and Output
- Coding Style and Conventions
- Number data types: int, double
- The Math Library
- Methods
- PA1 Tips

The Console

- Before graphical user interfaces, computers only had a text input and output system
  - Windows: cmd
  - Mac: terminal
- PowerShell is an advanced console by Microsoft

- There's a command for every kind of point and click action we have in a modern GUI
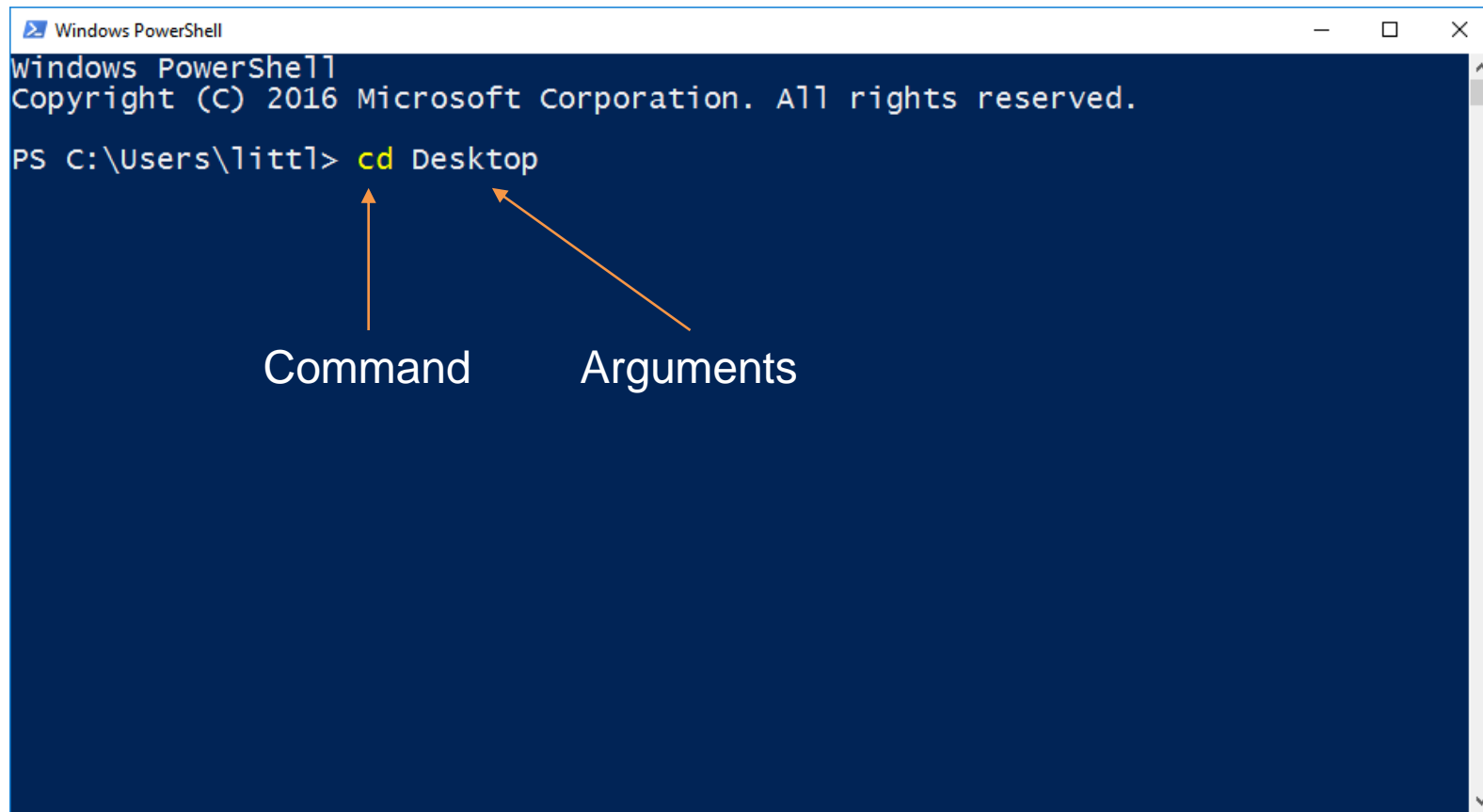
PowerShell Basics

PowerShell Basics

PowerShell Basics

Current Folder

PowerShell Basics

- **ls** (or **dir**) shows the contents of the current folder

PowerShell Basics

- **cd** changes the current folder:
  - **cd *subfolder*** will open up a subfolder
  - **cd *C:\path\to\some\folder*** *(Win) or* **cd *\/path/to/some/folder*** *(Mac)* will open up the folder given by this path
  - If any folder's name has a space in it, then the whole folder name or path must be surrounded by single brackets, eg: cd 'BME 121 Stuff' or cd 'C:\Users\My Files\School'

PowerShell Basics

- There are two special "directories" that are reserved folder names:
  - . (single dot) is the current folder
  - .. (double dot) is the parent folder of any folder
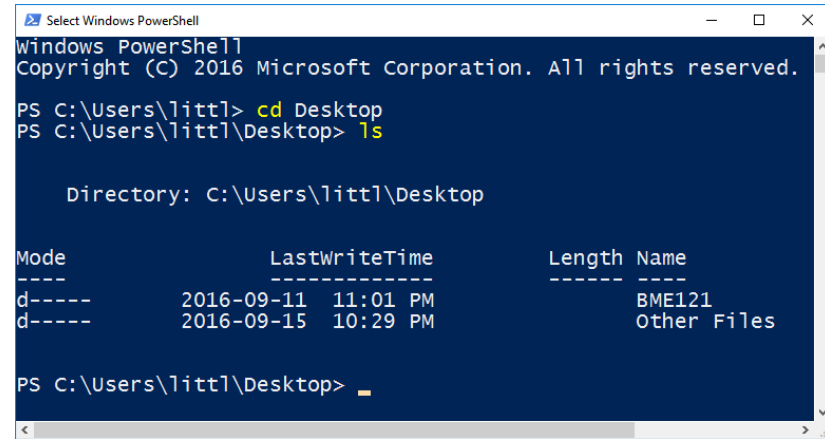
Practice 1

- Using File Explorer (Win) or Finder (Mac), open up your Desktop folder

- Using PowerShell, navigate to your Desktop folder using **cd** command then:
  - Use command **ls** to list the contents of your Desktop, see that the contents match the visual output from your GUI tool
  - Navigate to another folder of your choice using **cd**, navigate to the same folder using the GUI tool, and check that the contents listed by **ls** matches the visual output
  - Practice navigating into and out of subfolders using **cd *foldername*** and **cd ..**
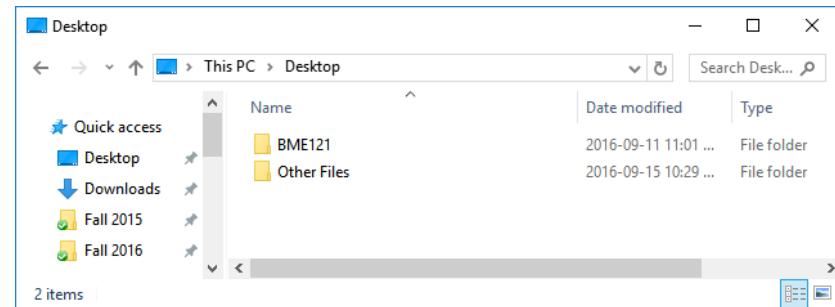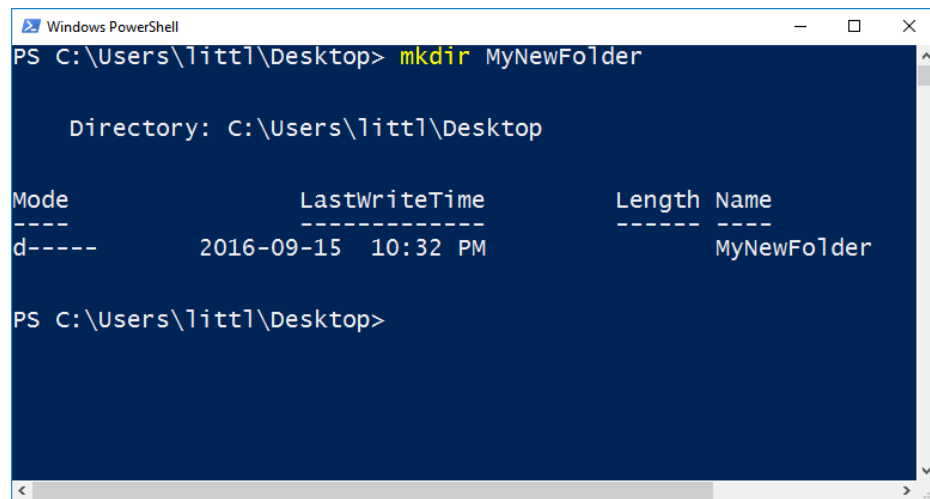
PowerShell Basics

- **mkdir *subfoldername*** creates a new subfolder with the name you provide as the subfoldername, in the Current Folder

- Like the **cd** command, if you want to create a folder with a name that has spaces in it, you must surround the entire folder name with single quotes
  - mkdir 'My Folder'
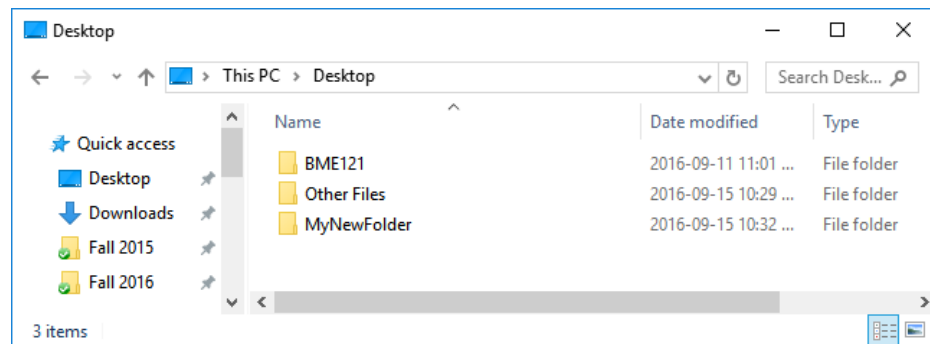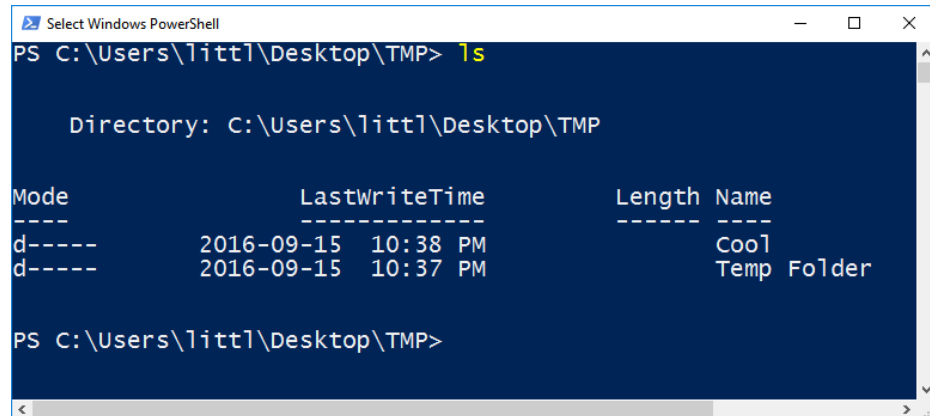
Practice 2

- Using a combination of mkdir and cd commands, create the following folders:

- In your Desktop folder, create a folder called TMP

- In TMP create a folder called Temp Folder

- In TMP create another folder called Cool

- Check that these folders exist using **ls** and your GUI tools
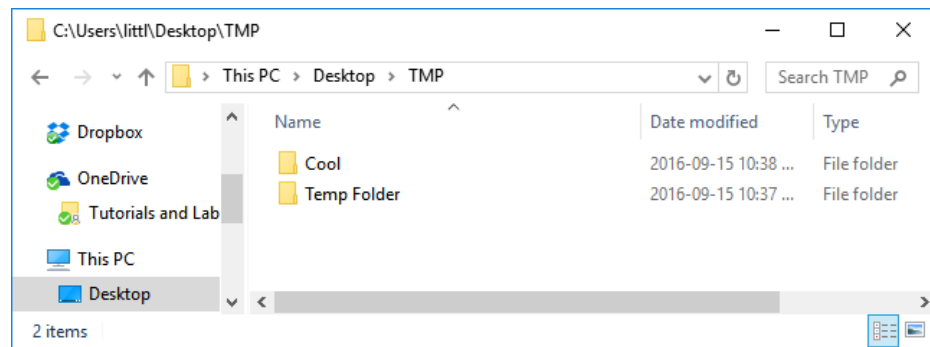
PowerShell Basics
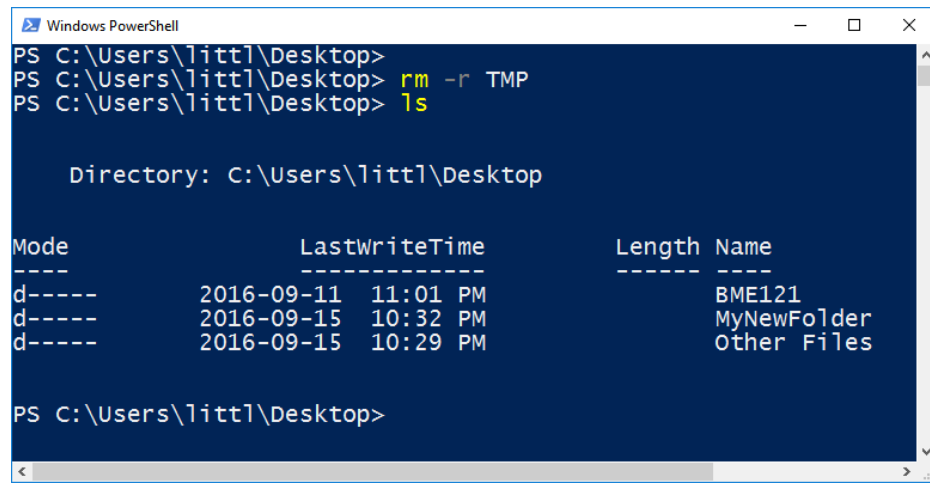
- **rm filename** deletes the file within the current folder, note that this won't ask you whether you want to delete or not, and once entered it is irreversible
- **rmdir subfolder** deletes the subfolder, but only if it is empty
- **rm -r subfolder** deletes the subfolder and any folder or files inside it

- **clear** will wipe the screen clean, which helps if your screen gets cluttered

- There's many more commands, Prof Freeman has a list of them in Lectures on OneDrive. You can also Google the commands!

Practice 3

- Clear the screen!
- **cd** to the TMP folder, navigate to TMP as well on your GUI tool, then remove the Cool folder using **rmdir** command. You should see it disappear in your GUI tool.
- **cd** to the Desktop folder, then remove both Temp Folder and TMP by using **rm -r** command. See that it is gone in your GUI tool as well.

Managing BME 121 Code Projects

- The compiler requires that you put each project into its own folder
- Use mkdir to create folders for each project, with names such as:
  - WA#, PA#, LAB#, TUT#, CH1Q1, etc
  - Feel free to put these in subfolders
- In empty project folders, you'll need to either:
  - Use dotnet new, dotnet restore, and dotnet run to create a brand new project, or
  - Copy the files from an existing project from another folder
- Customize the program.cs file from OneDrive with your name and uw ID, and keep a copy of the customized file as a starting template for most of the assignments in this course.
- Rename program.cs in each assignment to the assignment name: eg WA1.cs, PA1.cs, etc

Basic Structure of a Program

```
using System;
using static System.Console;
namespace Bme121.Xxx
{
    static partial class Program
    {
        static void Main()
        {
            // Your Code Here
        }
    }
}
```

Imports code for us to use, such as WriteLine()

Provides a context for all the named information in your program, eg variable names, class names, etc, which separates one Program from another.

Make sure to update "Xxx" to WA1, PA1, etc

Basic Structure of a Program

```csharp
using System;
using static System.Console;
namespace Bme121.Xxx
{
    static partial class Program
    {
        static void Main()
        {
            // Your Code Here
        }
    }
}
```

A Class groups up a number of methods and variables, and names the group (in this case Program)

A Method groups up a number of lines of code, and gives it a name.

Main() is a special method, it is the starting point of any C# program.
All C# programs must have exactly 1 Main method in some class.

Code Comments

- Comments are syntax features which allow us to write short notes or full sentences in English to let other programmers know something about our code. All comments are ignored by the compiler.

- Two ways:

- Single line comments:

```
// begins with double slash
```

- Block comments:

```
/* begins with slash star, and
* can go on for many lines until a star slash
*/
```

Code Comments

- Use this to jot notes!

- Write comments to help others with an equal programming ability as you to understand what your code does. This is expected for most assignments.
- Typically we put comments above or to the right of the code that it describes.

- In the long term (coop & career), write comments to communicate things that are not obvious by reading your code.
  - Companies have different standards and opinions on how much commenting is required or desired.

Program Variables

- A variable is a chunk of memory space reserved by a program, each used to store some information.
- Creating a variable: `type variableName;`
  - The variableName is the label we give to this memory space.
  - The type tells the computer what kind of information we are storing. This is required by the computer to allocate the required number of bits for each kind of information to a program.
- Example:

  ```
  string firstWord;
  string secondWord;
  int counter;
  ```

- firstWord and secondWord are two separate memory spaces, and both store a string.
- counter is a memory space that stores whole numbers.

Program Variables

- Storing information in a variable: `variable` `=` *`value`*`;`
- The = operator copies the value on the right hand side into the variable on the left hand side, eg:

  ```
  someString = "some words";
  someInteger = 5 + 5; // the value can be a computed result
  ```

- Reading/retrieving information stored in a variable: (have the variable name there)

  ```
  Console.WriteLine(someString);
  otherInteger = 5 + someInteger;
  otherString = someString; // copies the value
  ```

- A variable must have stored some value before it is read the first time. We call this **initializing** a variable (setting it's initial value).

Displaying Content to Screen

- Console.Write() method will display a string to the screen:

  ```
  Write("Some Words");
  ```

- The string can also have numbered placeholders for additional strings:

  ```
  Write("Words {0}, {1}", someString, someOtherString);
  ```

  - The placeholders are denoted by a pair of curly brackets, with an integer ID starting at 0.
  - The string from the first variable substitutes {0}, the second variable substitutes {1} and so on.
  - The number of placeholders must match the number of variables (or additional strings).
  - The string, and each variable is separated by a comma.

- WriteLine() does what Write() does and moves the printing cursor to the next line (aka presses enter key).

Obtaining Text Interactively From The User

- Console.ReadLine() obtains a line of text from the user.
  - Technically anything the user types until they press enter key, even if this shows up as multiple lines on the screen.
  - Returns the text as a string

```
// obtains some text and stores it in a freshly created
variable called userInput
string userInput = Console.ReadLine();


// obtains some more text and replaces what was stored in the
variable
userInput = Console.ReadLine();
```

Some String Methods

- + operator will join two strings together:
  - "hello" + "world"; // results in "helloworld"
- someString.ToUpper() returns a copy of the string with all alphabet characters in upper case
- someString.ToLower() returns a copy of the string with all alphabet characters in lower case
- someString.Trim() removes all leading and trailing spaces or tabs in the stored text.

- Many more: https://msdn.microsoft.com/en-us/library/system.string_methods(v=vs.110).aspx

Coding Style and Conventions

- Similar to the idea of essay citation styles (MLA, APA, Chicago, etc), coding has its own styles and conventions covering everything from how variables should be named to how the code should be formatted.

- Most companies have their own styles and conventions for each programming language.

- Microsoft defines a style for C#:

- Reference: https://msdn.microsoft.com/en-us/library/ff926074.aspx

C# Coding Style – Naming

- Variables - Lower case first letter and Camel Case for each additional word:
  - speedOfTheTrain, xPosition, zDisplacement
- Methods – Upper case first letter and Camel Case for each additional word:
  - ComputeAcceleration(), GetPosition(), MoveObjectOnZAxis(), BeGrumpy()

- Variable and method names must fully describe what the variable stores / method does. This helps other engineers and programmers understand your code!

C# Coding Style – Spacing

```
namespace Bme121.CodeFormatting
{
    static partial class Program
    {
        static void Main()
        {
            // Your Code Here
            string word = "Cat";
        }
    }
}
```

- Curly brackets for a namespace, class, method, etc must line up vertically.
- Content between each pair of curly braces is indented by 4 spaces or 1 tab.
- Good spacing helps you read code faster!

Reader Cat not impressed
by author shenanigans
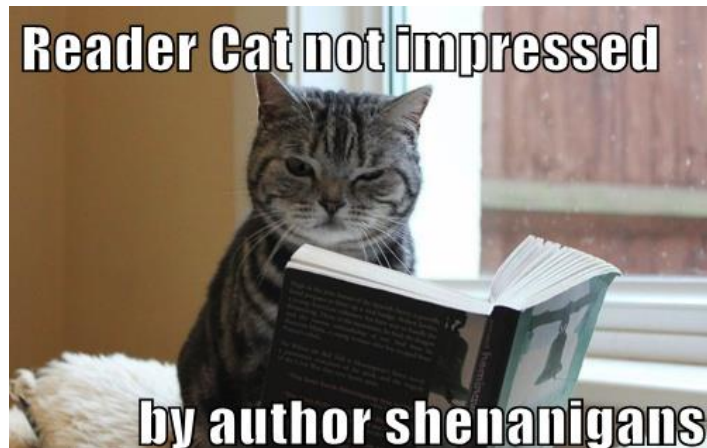
Practice 4

- Create a console program which asks the user for their first and last names, then prints out an award certificate similar to this screenshot:

- Requirements:

  - Good variable names

  - Perfect code formatting

  - Comments describing step by step what your code does

  - Remove any leading and trailing white spaces from the user's input


```
Lab2P4 Program
Enter your first name:      Jeff
Enter your last name: Luo

##############################
  Best Pokemon Trainer Award
  Jeff Luo
##############################
PS C:\Users\littl\Desktop\BME121\LAB2P4>
```

30 # characters

Number Types

- int: used to store whole numbers
- Limitations:
  - Range: -2,147,483,648 to 2,147,483,647

int x = 5;

- double: used to store real numbers
- Limitations:
  - Range: $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$
  - Precision: 15 to 16 digits max
- No default value
- Special values: `Double.NaN`, `Double.PositiveInfinity`, `Double.NegativeInfinity`

double literals need to either end in D or have a dot:

double x = 3D;

double y = 3.0;

double z = 32.123;

Integer Math Operators

- Addition: +
- Subtraction: -
- Multiplication: *
- (Quotient of a) division: /
- Remainder (of a division): %

- Brackets: ( )
- Code obeys Math's order of precedence when calculating numbers (% treated the same as * and /):
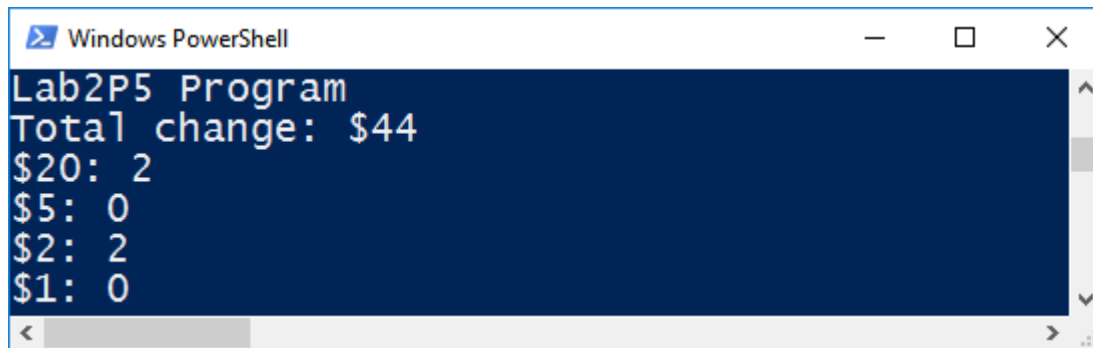
```
WriteLine(5 + 2); // prints 7
WriteLine(5 - 2); // prints 3
WriteLine(5 * 2); // prints 10
WriteLine(5 / 2); // prints 2
WriteLine(5 % 2); // prints 1
```

Called Modulus

Note: numbers are automatically converted to strings for WriteLine();

```
WriteLine(5 + 2 * 3 / (8 - 2) % 9); // prints 6
```

Practice 5

- A) Using the integer data type, write a short program which calculates the change that should be given to a user for a charge of $56 where the user pays $100 in cash. Specifically, display the number of bills and coins in each Canadian denomination that should be given to the user:

```
Lab2P5 Program
Total change: $44
$20: 2
$5: 0
$2: 2
$1: 0
```

- B) Adjust the program for a charge of $72, which should require 1 of each denomination

Double Math Operators

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Brackets: ()
- Also obeys math's order of precedence when calculating numbers.

Division by Zero

- In math, x ÷ 0 has no meaning as there is no number which, multiplied by 0 gives x (assuming x ≠ 0). In programming, there's a few outcomes.
- For integer division, the program will not compile if it is a literal zero:
  - `Console.WriteLine(5 / 0);`
- The program will compile but will crash if it is a variable storing zero:
  - `int x = 0;`
  - `Console.WriteLine(5 / x);`
- For double division, the result is either `Double.PositiveInfinity` or `Double.NegativeInfinity` depending on the sign of the calculation.
  - `Console.WriteLine(5D / 0.0);   // positive infinity`
  - `Console.WriteLine(-5D / 0.0);  // negative infinity`

C# Math Library

- The Console library provided us with methods which displays and retrieves text from the console. The Math library provides us with methods for common mathematical calculations, and scientific values:
  - y = Math.Sqrt(x); // calculates the square root of x and stores it in y
  - Math.Pow(x, n); // calculates x^n
  - Math.Min(x, y); // calculates and returns the minimum of x and y
  - Math.Sin(x); // calculates the sine of angle x in the unit of radians
  - Math.PI; // stores the constant π at 20 digits: 3.14159265358979323846
  - Math.E; // stores the constant e at 19 digits: 2.7182818284590452354
- Many more: https://msdn.microsoft.com/en-us/library/system.math(v=vs.110).aspx

Practice 6
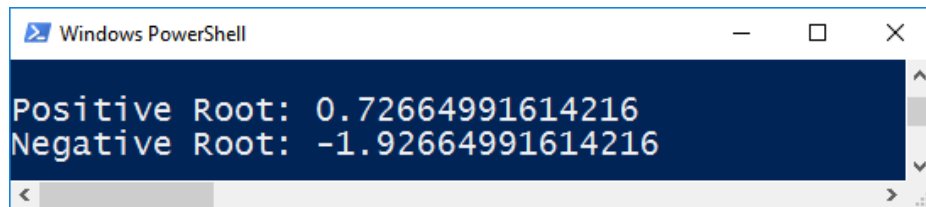
- Using the double data type, compute the roots of this quadratic equation:

$$y = 5x^2 + 6x - 7$$

- It will be helpful to compute the positive and negative roots separately as 2 equations:

- $x_+ = \dfrac{-b+\sqrt{b^2-4ac}}{2a}$

- $x_- = \dfrac{-b-\sqrt{b^2-4ac}}{2a}$


```
Windows PowerShell                                    —    □    ✕

Positive Root: 0.72664991614216
Negative Root: -1.92664991614216
```

C# Methods

- In Math, a function specifies a relationship between inputs and outputs:

  - f(x) = 5x + 7
  - g(x) = 8x – 9

INPUT x

FUNCTION f:

OUTPUT f(x)
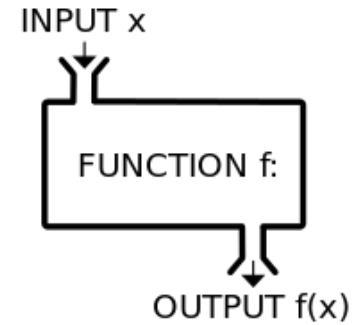
- When we have many functions and compose them together, we use the concept of value substitution to calculate the final result:
  - To calculate g(f(6)), we first calculate f(6) = 5(6) + 7 = 37, substituting 6 for x.
  - Then we substitute 37 for x in g(x) and calculate g(37) = 8(37) – 9 = 287

C# Methods

- In Programming, a method is like a Math function, but mechanizes the computation:

Eg: Paste the following after Main():

```
static int F(int x)
{
    return 5 * x + 7;
}
static int G(int x)
{
    return 8 * x - 9;
}
```

In Math:
f(x) = 5x + 7
g(x) = 8x - 9

Then add this in Main():

```
Console.WriteLine( G(F(6)) ); // displays 287
```

C# Methods

- Parts of a method:

```
<visibility modifiers> <return type> <name>(<parameters>)
{

        <code>

        <return statement>

}
Eg:
static int G(int x)
{

        return 8 * x - 9;

}
```

C# Methods

- Like Math substitution, when we call a method, the input variables are assigned the values that we give it:
  - When we call F(6), variable x in method F takes on value 6
  - We call this passing a value into the method
- The return statement passes a value out of the method:
  - When we call G(F(6)), the value returned by F(6) is temporarily stored in memory, then read and passed into G()

C# Methods

- Method evaluation is always inner before outer:
  - `Console.WriteLine(G(F(6))); // displays 287`
  - F(6) is executed first, returning a value which is passed into G(), which then executes and returns a value that's passed to WriteLine()
- Each call to the same method runs the method from scratch (the results are not stored by default):
  - Console.WriteLine(F(6) + F(6)); // runs F(6) twice, then adds them up and displays the final result
- To be efficient, we can run methods once, store the results, then compute the next part of the equation:
  - double fSix = F(6);
  - Console.WriteLine(fSix + fSix);

C# Methods

• We can also have methods that aren't mathematical. All methods use the idea of substitution:

```
static void DisplayProgramTitle(string program, string name) {
    WriteLine("BME 121 " + program);
    WriteLine("by {0}" + name);
}
```

Note: the void type denotes that a method does not return anything, only in such methods can the return statement be omitted. See void Main().

C# Methods

- It is very common to package formulas or code into methods:

```csharp
static double PositiveQuadraticRoot(double a, double b, double c)
{
    double bSquared4AC = Math.Pow(b, 2) - 4 * a * c;
    double dividend = (- b + Math.Sqrt(bSquared4AC));
    return dividend / (2 * a);
}
```

- When called by Main, these methods help Main accomplish its objectives (thus called helper methods).

- Remember: unlike the example of F() and G() in this slide deck, your methods should always have descriptive names!

C# Methods – Coding Style

- For methods, we provide comments as a form of in-code documentation to describe what the method does, the purpose of each variable, the value returned by the method (if applicable), and any special conditions:

```
/* Computes the positive root of a quadratic equation
Inputs:
        a – the constant multiplicand of the x^2 term, cannot be zero
        b – the constant multiplicand of the x term
        c – the constant of the equation
Returns:
        the positive root
*/
static double PositiveQuadraticRoot(double a, double b, double c)
```

Why?

- In online code documentation, you'll see the same kind of information presented: https://msdn.microsoft.com/en-us/library/system.math.pow(v=vs.110).aspx

PA1 Tips

- Choose good variable and method names
- Add comment lines above each variable to indicate the unit of the value stored in the variable
- Document your helper method with comments, as we have shown here
- Break down the equation into a number of computation steps to make the code easier to read and understand
- Reduce re-computation by saving the results of repeated (partial) calculations
- Double check all assignment and submission requirements
- Make sure the namespace is `Bme121.PA1`
- Make sure the file you submit is called PA1.cs
- There's one more thing you need to learn for PA1, taught next week, but get it started!