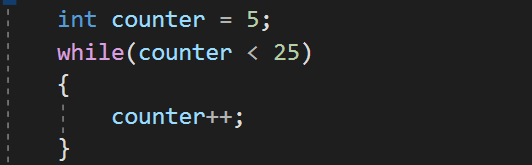
The purpose of this worksheet is to focus down, step by step, on the problem-solving process and offer you a chance to flex your critical thinking muscles. This worksheet will outline the process for breaking down problems into their most simple elements.

Let’s begin with working with some simple for-loops. Changing and tweaking them to get them to produce different results.

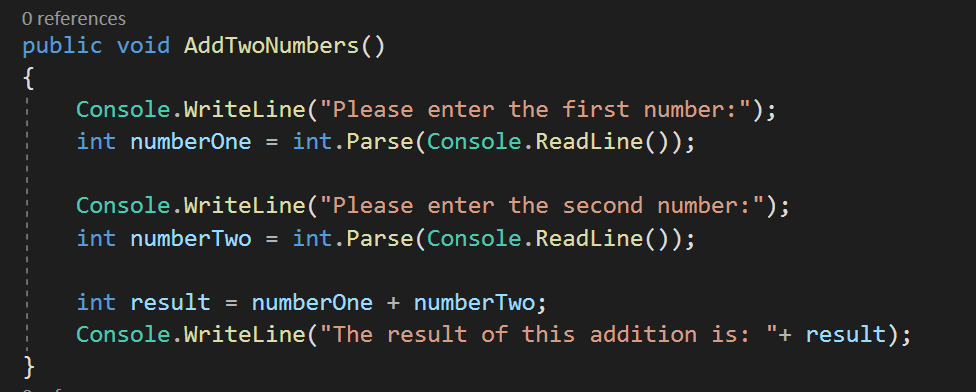
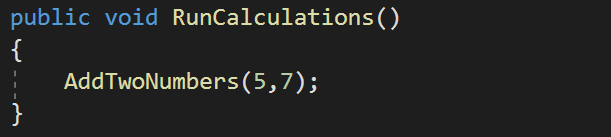
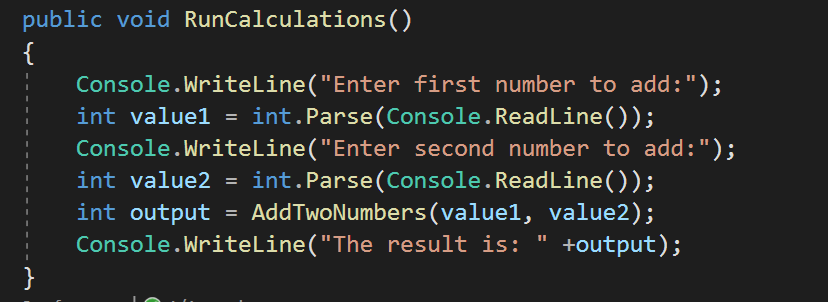
Fun with For-Loops

1. Write a for loop that will run 5 times.
2. Modify the loop to print out every incrementation of the for-loop’s counter. (Usually “i”)
   1. I.e 0,1,2,3,4
3. Using the same loop, change it to now print backwards from 9.
   1. 4,3,2,1,0
4. Write a loop that will run as many times as a user wants.
   1. Change the loop to run “Forward” again.
   2. Take in user input and use the value to run the for loop.
   3. Valuable research terms:
      1. “How do I capture user input in C#”
      2. “How do I convert a string into an int in C#”
5. We are now going to modify this loop to run as many times as there is letters in a string.
   1. Declare a string above the for loop with a value of “Hello World”.
   2. Your loop should run 11 times for each letter in the string.
      1. Note: using “ i < 11” is not what we are looking for. I should be based off the string itself. So that we could have this working for any string, regardless of how many characters is present.
   3. Your loop should now output: 0,1,2,3,4,5,6,7,8,9,10
   4. Valuable research terms:
      1. “How do I get the number of characters in a string C#”
6. Using the same loop from step 4. Let's modify the logic to instead print out every letter of our string one at a time.
   1. I.e “H,e,l,l,o, ,W,o,r,l,d”
   2. Valuable research terms:
      1. “How to print one letter from a string”
7. Using the same loop from step 6. Modify it to print out every OTHER letter.
   1. I.e “H,l,o,W,r,d”
8. Let's take this same loop and modify it to only print out if the current index is divisiable by 3.
   1. I.e “H,l,W,l”
   2. Valuable research terms:
      1. “How to find if a value is PERFECTLY divisible by another number”

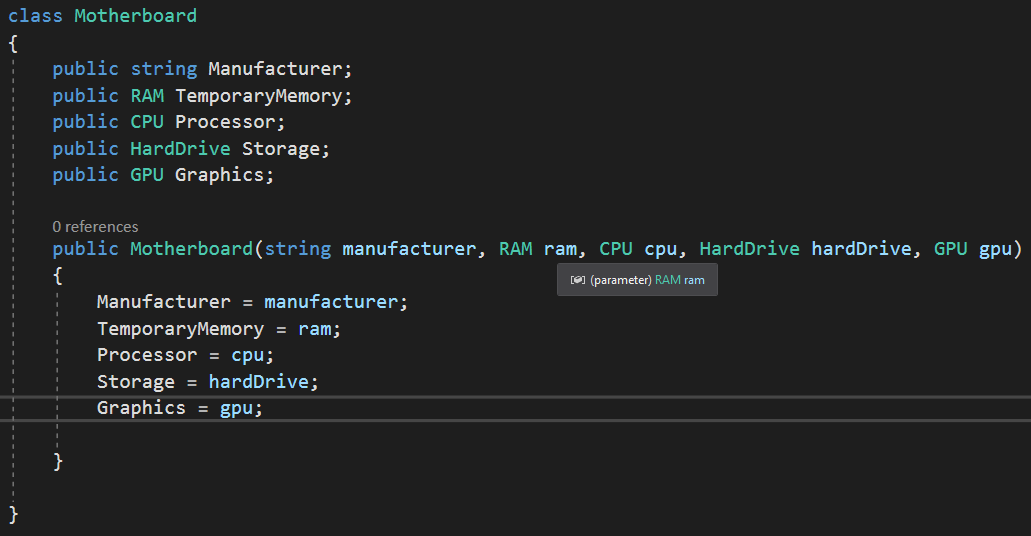
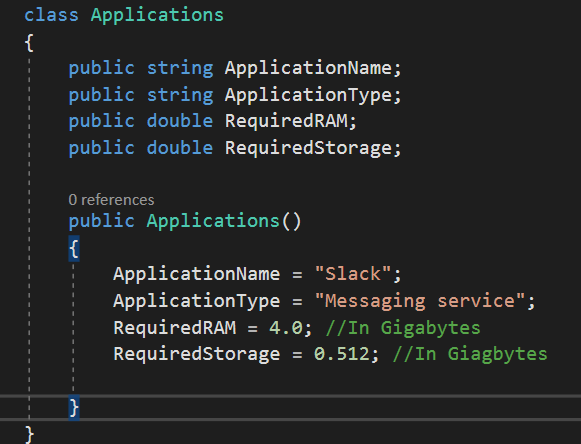
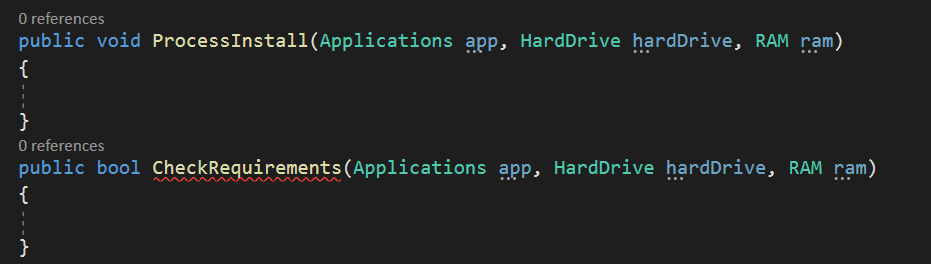
Fun with While-Loops

1. Write the following while loop.
   1. 
   2. Questions to think about:
      1. How many times will this loop run?
      2. What is the difference between this while loop and a for loop?
2. Let's now modify this loop to take in user input for our counter instead of hard coding a value. Our user should be able to enter any number they choose to start off our counter variable.
   1. You are going to need to capture the input from the user then turn it into an integer.
   2. Questions to think about:
      1. What is something you think could go wrong with this set up?
      2. How do you think we could fix this problem?
      3. How many possible times do you think this loop will iterate?
      4. What if they enter a negative number?
3. Let's continue to tweak this loop. Let's change the condition for the loop to check to see if our counter is not equal to 42. If the value is not equal to 42 then we should re-prompt the user until they enter the correct number. Just for clarification for the user; we should add some outputs to our logic. We should both ask our user to input the initial value with something like “Please enter the number 42”. And another output for if the user enters it wrong.
   1. Questions to think about:
      1. How many times is this loop guaranteed to run?
      2. Is there potentially another type of while loop we could use for this logic?

Fun with Methods

1. Let's create class to house our methods that we will be creating in this section and call it “CalculatorMadness”. No need for any member variables or constructor.
2. Let’s next create a method inside our new class like this:
3. This method as it is currently is relatively dynamic. Lets make this a bit more versatile though. Let’s change this method to instead take in numberOne and numberTwo as parameters.  
   1. When we change the method to take in the values instead. We no longer need to ask for user input in this method. Rather the variables will be assigned values from where the method is called.
4. Let’s create another method in our class called “RunCalculations”. Inside this method we will call our “AddTwoNumbers” method and pass it the values it is looking for.
   1. 
   2. When we move the variables to parameters, this allows the method to be used in more ways now. It isn't specifically based on user input anymore; it can now be used any time you want to add two values together.
5. We still have more work to do on this method. For example, this method currently just outputs the result to the console. Which means we can't do anything further with our result. Let's change our AddTwoNumbers to instead return our result back to where it was called.
   1. You will need to change the return type of the method to now reflect the new return type.
   2. Now that we have made this method take in two parameters and return the result. We have made this method much more flexible. It can be applied in a much larger spectrum of situations. We can still call the method using user input if we want. But now we can use it for any and all two-number addition.
   3. 
      1. Example of passing user entered values into out add method^
6. Using only the Add method, we will call this same method multiple times to add more numbers together than just two.
   1. First, we will want to add 8 + 40. Then we will want to add the result of that to 200 + 50.
      1. (8+40) + (200+50)
   2. Capture the result and print it to the console.
      1. You cannot use the + operator in your RunCalculations method.
7. Next, we will create 4 new methods very similar to our Add method.
   1. Create a method to subtract one number from another number.
   2. Create a method to multiply two numbers.
   3. Create a method to divide one number by another number.
8. Using only the RunCalculations method and math methods you created. Have your logic solve the following math problem:
   1. 6+5-40\*35/4+22
   2. Your output should be –335
      1. Keep in mind order of operations!
      2. 6 + 5 - ((40\*35)/4) + 22

Fun With Classes

1. In this section we are going to build a computer! (Kind of)
2. Let's start by creating a “Computer” class. This will be our most top-level object. This will be the Case that we will fill with all our other computer parts.
3. Next let's create classes for all the components of our computer. We will need:
   1. GPU class (Graphics processing Unit)
      1. This handles the graphics operations for the computer (primarily for 3D calculations).
      2. Member variables:
         1. String Manufacturer;
         2. Double EffectiveMemory;
   2. CPU class (Central processing unit)
      1. The CPU performs basic arithmetic, logic, controlling, and input/output operations specified by the instructions in the program
      2. Member variables:
         1. String Manufacturer;
         2. String Name;
   3. HardDrive class
      1. Stores any and all content for your computer. Persistent data.
      2. Member variables:
         1. Double TotalStorage;
         2. Double Available Storage;
   4. RAM class (Random-access memory)
      1. Essentially your device's short-term memory. It temporarily stores (remembers) everything that runs on your PC, like all the services in Windows, your web browser, your image editing tool, or the game you're playing.
      2. Member variables:
         1. Double TotalGigabytes;
         2. String Brand;
   5. Motherboard class
      1. The Motherboard itself is a printed circuit board that allows the CPU, RAM, and all other computer hardware components to communicate with each other.
      2. Member variables:
         1. String Manufacturer;
         2. CPU Processor;
         3. RAM TemporaryMemory;
         4. HardDrive Storage;
         5. GPU Graphics;
4. Next create constructors for all of the classes. We want to assign all of the member variables with the constructor during instantiation. We want to make sure to pass all the values into the constructor. We want these classes to be built into any different kind of whatever. No hard-coded values in our constructor.
   1. Example for Motherboard:
   2. 
5. Since we have set up our classes this way, we can supply each class with any values we want upon instantiation.
6. Now let's imagine that we wanted to add new programs to our computer. Like Microsoft Word, Slack, Visual Studio, etc. We want our computer to be able to install the programs and store them in our hard drive. In a List of Applications.
7. Let’s create another class. An Applications class and we will hard code in some values for now:
   1. 
   2. We are going to “Install” this application on our computer.
8. Now let's create some methods!
   1. Before we start let's add a new member variable to our HardDrive class.
      1. Public List<Appliactions> ApplicationsInHardDrive;
   2. In our motherboard class create a method called “InstallApplication”. This method should take in an Applications object as a parameter.
   3. In the method. Let's add the application to our ApplicationsInHardDrive variable on the HardDrive.
      1. Dot notation will be very useful for this.
   4. This is a start; however, we should probably include some logic to make sure this application can be installed by our system.
   5. Let’s include an if statement that checks to see if our system has the necissary requirements to hand this application.
      1. We should check to see if our RAM is greater than the RequiredRAM of the application and check to see if our hard drive’s available storage is greater than the RequiredStorage of the application.
      2. If we have enough specs for the application, only then should we add the application to our List of programs.
9. Let's take a moment to think about the responsibilities of these Objects. Currently we have the Motherboard handling most of our logic. When really the processor should be handling most of the logic. The Motherboard is mainly the means for these components to communicate with each other.
   1. Let's move the logic for checking system requirements to the processor. As well as the actual install to the harddrive.
   2. 
   3. Fill in the logic for the two methods.
10. Great! Now let’s talk inheritance. Currently we only have one type for application. Really there should be multiple types of applications.
    1. Let's turn our applications class into an abstract class.
    2. Let’s add two other classes, “Games” class and a “TextEditor” class. Both need to inherit from Applications class.
    3. The TextEditor class doesn’t need any additional variables on it, however, the Game class will need a new variable for “RequiredEffectiveMoemory”
    4. Make the necessary adjustments to the methods
       1. We now need to take into consideration if our systems graphics can handle the game or not.
11. Last! Let's build our computer!
    1. We need to build all of the computer components via their instantiation and put them in their proper spot!
    2. We should try to build some applications both games and not and try to “Install” them to our computer!