**hName: Jason Truong**

For each of the below questions, write a short sentence or two to express (in your own words) your answer. Keep the answers short, but use complete, correct, English sentences.

If it helps to clarify the questions, feel free to mentally prefix all the questions with the phrase "According to the video…"

* After you’ve watched all the videos, please answer this question:  
  Of all the videos that you watched, if you could pick one video to be re-recorded by the instructor outside of class which would you choose? Why?  
  (Keep in mind the recording outside of class will omit any pauses from the instructor answering student questions, have less hemming and hawing, etc, and generally be more concise)

|  |
| --- |
| I need to turn up the videos rather loudly to hear, and Video 07 was an attack to the ears. I had to rip the headphones off.  Should've braced for it. |

**VIDEO: Basic Parameters**

* For an instance method, what do I need to have (in my program) before I have can have my program call the instance method?

|  |
| --- |
| For instance methods, you need to have it as a call-able object to be called on initiation when designated. |

* Copy the example from the video (up to the 1:30 mark) of defining a class named Demo with a method named Triple, along with an example of calling that method from within **main**. (At this point the Triple method doesn’t actually do anything)

|  |
| --- |
| class Demo  {  public void Triple()  {  }  }  class Program  {  static void Main(string[] args)  {  Demo d = new Demo();  d.Triple();  } |

* How do you add a parameter to the Triple method’s definition?  
  How do you pass the value 7 into the method call for Triple in **main**?

|  |
| --- |
| You create a variable parameter for it to be passed through in the original method.  To pass 7 through the parameter, you would have to set up the object and pass 7 through the object as you called it. |

* Does the name of the parameter (in the method’s definition) have any connection to any variables declare in the **main** function?

|  |
| --- |
| As its literal name? No, it acts synonymously as a name for the method's parameter variable, as its injected a variable from the main class. No it doesn't act as a mnemonic. |

* Briefly, intuitively describe what the “Call stack” is, and what “stack frames” are.

|  |
| --- |
| It acts as the code's own "Order of Operations". A 'stack frame' are the operations in order for the 'call stack'. |

* Will changing the value of **x** in the **Triple** method ever affect the value of **num** in the **main** method?

|  |
| --- |
| No, it's an entirely different 'local copy' for Triple. |

* If you want to list out multiple parameters, how do you separate them? How would you specify that you want an integer parameter named **x**, and a double named **y**.

|  |
| --- |
| With a comma.  public void Triple(int x, double y) |

**VIDEO: Basic Return Values**

* Briefly, intuitively describe what “Default parameters” are used to accomplish.

|  |
| --- |
| When the parameter-necessary code is called without a valid parameter, it has a 'safety net' for the code to fall back on. |

* Copy down the example from the video that has the Triple method taking it’s parameter, tripling it, and then sending that new value back:

|  |
| --- |
| namespace ConsoleApplication1  {  class Demo  {  public int Triple(int x)  {  int result = x\*3;  result result;  }  }  } |

* When Visual Studio tells you that your method has the “Not all code paths return a value” error, what problem does your code have and how do you fix it?  
  (Explain this clearly **in your own words**)

|  |
| --- |
| Return something; anything, so it appeases the compiler. Despite it never being returned. |

* If a method produces a return value are you required to make use of that return value in, say, **main**?

|  |
| --- |
| Nope. It returned a value, but you don't need to use it. |

**VIDEO: 02\_String\_Compare.mp4: Q + A: String Comparison with ==, String.Compare**

* During the execution of the following code snippet, how many string objects will be created? Once the snippet has finished, what object will **input** refer to? What object will **input2** refer to?  
  string input= “ok”;  
  string input2= “ok”;

|  |
| --- |
| Two different string objects both holding "ok".  input and input2 will refer to the same string block. |

* During the execution of the following code snippet, how many string objects will be created? Once the snippet has finished, what object will **input** refer to? What object will **input2** refer to?  
  string input= (string)“ok”.Clone();  
  string input2= (string)“ok”.Clone();

|  |
| --- |
| Two string objects both holding "ok", but this time, not referring to the same block of memory stored.  So it makes a brand-new copy for that object accurately. |

* In C#,what does the double-equals ( == ) operator do? Continuing the code snippet from the prior question, will the following evaluate to true or false?  
  if( input == input2 )

|  |
| --- |
| It checks if the content are both the same in both letterings and case. |

* What are the three possible results that the String.Compare( string1, string2, true ) method might produce? What does each value mean?

|  |
| --- |
| It would produce either 0; if the same, -1; if first goes first, 1; if the second value goes first.  string1 is the first value  string2 is the second value  true/false is the conditional for the check being case-sensitive. |

* What does the ASCII standard define? How does Unicode differ from ASCII?

|  |
| --- |
| ASCII is the typography equivalent to numerics.  Unicode is an expanded ASCII that allows non-english letters - 'super-set'. |

* How can you control whether String.Compare is comparing the two strings in a case-sensitive manner or a case IN-sensitive manner?

|  |
| --- |
| Change the final variable of 'true/false'.  True = case-sensitive  False = case-insensitive |

**VIDEO: 04\_Basic\_Array\_Stuff.mp4: Basic Array of Primitive Types**

* Give some examples (2-3) of C#’s “primitive data types” or “simple data types”.  
  What do all primitive/simple types have in common?

|  |
| --- |
| Primitive/Simple types are ints, ulong, uint.  Simple data-types take part in the call stack with the main, and does not need to request separate or special storage space for it to exist. |

* Is an array a simple type? How do we know?

|  |
| --- |
| An array is not a simple type because it has to require extra space for itself to exist. |

* Briefly, intuitively, what does **the heap** contain?

|  |
| --- |
| The heap is a separate 'stack' but stores whatever is initialized as 'new'.  But it doesn't actually go into the Stack, because it is something created and exists outside of the stack's existence. And as long as its needed, called, or constant, it is a new. |

* In terms of the length of time that variables and objects exist, how are the stack and the heap different?

|  |
| --- |
| Heaps are 'permanent' variables whle the stack 'forgets' the steps before it to make progress forward. |

* When C# creates an array of integers, what value will C# fill the array in with? What value is used to fill in an array of doubles?

|  |
| --- |
| Integers: 0  Doubles: 0.0 |

* What is a good way to think about your computer’s memory (about your computer’s RAM)?

|  |
| --- |
| Memory is a giant array of bytes |

* When the instructor talks about something being located at memory address 70,000 where is that thing located in RAM (explain this using the intuitive model you described in the previous question)?

|  |
| --- |
| 70,000th spot in the 'giant array of bytes' previously mentioned. But each slot of the array are actually 4 bytes in the heap, so when the 70,000th spot in the array are actually '28,000'.  But we'll never know. |

* What is the value of nums at minute 15? What does this value represent / what does it mean?

|  |
| --- |
| 70,000.  It represents its place from the stack, into a length of heap equal to it's value. |

* Give a short C# snippet to demonstrate how you would put the value 20 into slot 0 in the array referred to by the **nums** variable.  
  Also give a short C# snippet to demonstrate how you would print out the value stored in slot 0 in the array referred to by the **nums** variable.

|  |
| --- |
| num[0] = 20;  It assigns the integer value in that space in the array.  Console.WriteLine(num[0]); |

* In C#, what is the index/address of the first element of all arrays?

|  |
| --- |
| The 0th spot in the array. |

* If there are five elements in the array, what is the highest valid index?  
  If there are **N** elements in the array, what is the highest valid index?

|  |
| --- |
| 4.  N-1. |

* In C#, if you create an array with 5 slots can you assign a value to slot 10?

|  |
| --- |
| No. Out of bounds. |

* In C#, can you change the size of an array? If not, then what can you do that’s equivalent to changing the size of the array?

|  |
| --- |
| No, not the array itself.  Take a new array, copy all the values in the previous array, and pretend the previous array doesn't exist.  Extinguish it from your memory. |

* Typically, when you resize an array, is it a good idea to increase the size of the array by only 1 new element? Why or why not?

|  |
| --- |
| You'd wanna double the size of the array, or just move it up a large number. Or else the program will just encumber the code as it keeps reiterating on itself. |

* Memory that you cannot use because you have no variables that refer to it (directly or indirectly) is called what?

|  |
| --- |
| Garbage memory |

* If C# (or Java) runs out of memory, what does it do? What is this process called?

|  |
| --- |
| It allocates all the space, then goes into the garbage memory, and if there is no space then promptly crashes.  Garbage Collection. |

* Why is it safe for the computer to jump directly to element 10 of an array (rather than searching through memory, slot by slot, for element 10)?

|  |
| --- |
| Because it knows when to start, and where the start is. So it is fully aware of how to jump to element 10 safely. |

* If you wanted to print how many elements are in an array, how could you do that? (Give a brief snippet of C# code)

|  |
| --- |
| Console.WriteLine(arrayIsThis.Length); |

* Why is it useful to choose the size of an array (when we first create/allocate the array) using a variable?

|  |
| --- |
| Because array's still count from the 0th spot as 1-element space. |

* Copy the example C# code that can print out all the elements of an array into the space below

|  |
| --- |
| for (int index = 0; index < nums.Length; index++)  {  Console.WriteLine("Slot {0} contains the value: {1}", index, nums[index]);  } |

* Briefly explain how your code in your answer to the previous question works.

|  |
| --- |
| The loop starts at 0, and ends at the digit that is the length of the array-1.  It takes the index of the for loop, and walks the length of the array, and pulls each value of the space in nums[index]. |

**VIDEO: 05\_How\_To\_Adjust\_Console\_Buffer.mp4: How To Adjust The Console's Buffer Size**

* How can you adjust the console’s buffer size in Microsoft Windows? Assuming that the console window is open, list each of the steps that you need to go through (first click here, then click this, etc, etc)

|  |
| --- |
| Click the icon at the top left of the console's window, then go into properties. Then while the properties windows is open, access it through the 'screen buffer size'. |

**VIDEO: 06\_Array\_Params.mp4: Arrays as Parameters (syntax)**

* When calling a method that accepts an array as a parameter, how do you print the name of the array?  
  (I.e., in main, how do you write out the name of the array)

|  |
| --- |
| With the example in the video.  Demo d = new Demo();  double[] vals = new double[7];  d.PrintArray( vals ); |

* In the example, how was the array parameter declared for the PrintArray method?  
  Do you need to include square brackets in the method’s definition?

|  |
| --- |
| public void PrintArray(double [] nums)  {  xxxx  }  Yes, you need to include the square brackets, those specify whether something is a single variable versus an array. |

* Copy over the entire definition for the Demo class and the PrintArray method (use the version that has only the array parameter)

|  |
| --- |
| class Demo  {  print void PrintArray(double[] nums, int x)  {  for (int i = 0; i < nums.Length; i++)  Console.WriteLine(nums[i]);  }  } |

**VIDEO: 07\_Pass\_By\_Reference\_Value.mp4: Arrays as Parameters (pass by reference vs. pass by value)**

* In the example in the video at the 2:00 minute mark, what value do both **vals** and **nums** have? What does this mean in terms of which actual block of memory each one is referring to?

|  |
| --- |
| 70,000.  You're passing a single reference to the one array in multiple places. |

* Because of your answer to the prior question, executing the C# code for nums[1] = 10; in the PrintArray method has what side effect?

|  |
| --- |
| It'll pass the number 10 in the 2nd slot of the array. |

* What is the name for passing a variable this way?

|  |
| --- |
| Passing by reference. |

* Fill in the blank: “When passing a simple type, you actually get a \_\_\_\_\_\_\_\_\_”

|  |
| --- |
| copy directly. |

* At the 5:00 minute mark in the video, if you change the value of y what effect does that have on the variable x?

|  |
| --- |
| The value of the y is changed.  The value of x does not change, only the 'copies' are changed. |

* What is the name for passing a variable this way?

|  |
| --- |
| passing by value. |

**VIDEO: 08\_Return\_Values.mp4: Arrays as return types**

* Copy down the example that demonstrates how to return an array from a new MakeArray method from the video (at the 1:30 mark)

|  |
| --- |
| public int[] Makearray (int size)  {  int[,] newNums = new int[size, size];  return newNums;  } |

* When returning an array, do you use square brackets?   
  Give a short C# snippet to demonstrate how one might return an array named newNums:

|  |
| --- |
| No. Just return the name like this: newNums; |

* Describe (briefly and intuitively) the two different types of multidimensional arrays that are available in C#. What is the name of the second type?

|  |
| --- |
| A matrix of memory.  Jagged array. |

* Copy down the example that demonstrates how to receive an array from the MakeArray method and assign it to a variable named differentName (around the 6:00 mark)

|  |
| --- |
| public int[] MakeArray(int size)  {  int[] newNums = newint[size];  return newNums;  } |

* Given the following code snippet, how many arrays are allocated? At the end of the snippet, which array does differentName refer to?  
  int[] differentName;  
  differentName = d.MakeArray(10);  
  differentName = d.MakeArray(10);

|  |
| --- |
| It creates two arrays, but disposes of the first. At the end of the array, it refers to the second differentName which is at a different address. |