**Name: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)

|  |
| --- |
| Nada, no mind, nothing, sorry. |

**VIDEO: While Loops**

* In a nutshell, how does the **while** loop execute?

|  |
| --- |
| It checks the block, does the criterions, comes back to the top, checks the while criterion, ccontinue until stated otherwise |

* What is typically true of a situation where you would want to use a while loop?  
  (At the outset/start of the execution of the loop, what does the while loop typically NOT know?)

|  |
| --- |
| Typical moments is when you are usure how often you need to dance through the loop |

* Around the 3:10 mark the instructor explains a good example of where you might use a while loop (one that’s much better than the loop that counts down which is used at the start). What is this example ,and why is it a good situation to use a while loop?

|  |
| --- |
| You continously ask the user for a set amount of actions, like you repititously badger them to typ in '1', and as soon as they stop the program stops. |

* Syntactically, do we need the opening and closing curly braces that follow the line with the word while on it? If you do NOT have curly braces then which lines are considered to be part of the while loop (i.e,. which lines will be repeated)?

|  |
| --- |
| The line that immediately follows it and not anything else |

**VIDEO: For Loops**

* A **for** loop is made for doing what?   
  Also – fill in the blank: “Knowing \_\_\_\_\_\_\_\_\_\_ is usually a good indicator that you should use a for loop”

|  |
| --- |
| how many times to execute a loop |

* Copy the example of the for loop that was show in the video here:

|  |
| --- |
| for (int i = 0; i < 10; i++) { Console.WriteLine(" i is: {0}", i); } |

* What does **i++** do?

|  |
| --- |
| i++ increases the for-loops internal counter |

* Inside the for loop, what symbol do you use to separate the initialization from the check from the increment?

|  |
| --- |
| semi;colons |

* When you arrive at the for loop (on line 25), how many times do you do the **int i = 0** part? Is it possible that the program might ever skip this part?

|  |
| --- |
| One time. if its pre-established that the variable i exists, it can just skip it entirely. |

* When you arrive at the for loop (on line 25), what is done after the the **int i = 0** part? What is the minimum number of times that this part will be done?

|  |
| --- |
| Check the variable, check the second criterion go through the code block, repeat steps 1-4 until criterions reach false.  Can even be done only once if the check only goes through once. |

* Next, the **for** loop will do the body of the loop? What is the body of the loop given in the example in the video?

|  |
| --- |
| The for loop will do the body of the loop if its |

* Next, the **for** loop will do what part of the loop?

|  |
| --- |
| Goes up an increment |

* If any point the check part is false, what does the program skip? What does it do next?

|  |
| --- |
| Skip its 'body' code and moves onto the next line after. |

**VIDEO: Do-While Loops**

* What is the major difference between a do-while loop and the other two types of loops that we looked at?

|  |
| --- |
| It will have a predisposed do statement(s) before actually going through into the next while loop following it. Then the while loop will act on the encapsulated statement encased in the 'do'. The while loop will continue until the while criterion will reach false, then it will stop the 'do'. |

* Do…while loops MUST have a semi-colon where?

|  |
| --- |
| At the end of the while statement. |

* In C# (and many other languages), what is **whitespace**? What implication does this definition have in terms of where we can put extra blank lines in our programs?

|  |
| --- |
| The space between codes when you press tab or space inbetween the code that doesn't actually affect the code. |

**VIDEO: Increment (++) and decrement (--) operators**

* Exactly how does **x--** work?

|  |
| --- |
| Stash a copy of the established variable and return it into the original expression, and instead of increasing the value by 1, it decreases it instead everytime the statement is passed |

* Give the example of C# code that illustrates this that was used in the video, and make sure to explain why it prints 10 and NOT 9.

|  |
| --- |
| Because it prints the stored statement then changes the value. |

* Exactly how does **--x** work?

|  |
| --- |
| It changes the value first, and then it store the new value. |

* Give the example of C# code that illustrates this that was used in the video, and make sure to explain why it prints 9 twice and NOT 10 then 9 (and then prints 8 twice instead of 9 and then 8, etc, etc).

|  |
| --- |
| Because it changes and stores the new value, then it prints it. Then it prints the next statement, but the value hasn't changed so it just duplicates it. Then it goes to loop and changes the value again. |

* In the following code snippet, why does it not matter whether use you i++ (as written) or ++i?  
  for( int i = 0; i < 10; **i++**)

|  |
| --- |
| It will completely change on if the value gets stored then edited, or edited then stored. |

**VIDEO: Compound operators (+=, \*=, etc)**

* In addition to C#, what other languages have the ++ and -- operators? Very briefly explain why.

|  |
| --- |
| Java, the decriment and increment feature was also featured in C++. All because of its parent code; C. |

* Does **++x++;** work?   
  (You only need a yes/no answer for this question, and then you need to remember this answer when writing up your own programs. )

|  |
| --- |
| No |

* Instead of writing out **x = x + 2;**, how can I write an equivalent statement using a compound operator?

|  |
| --- |
| x += 2 |

* List out the other compound operators used in the video, and which basic operation each one does:

|  |
| --- |
| \*= /= -= |

**VIDEO: Random Numbers: Seed Values**

* Can C# actually generate a truly random number? What is the brief, intuitive explanation that’s offered in the video?

|  |
| --- |
| Nah, C# can't cause it has a process that it does every time similarly to the last. |

* Copy down the line that creates a Random object from the example in the video.  
  If the parentheses are left empty, what is used as the “seed value”?

|  |
| --- |
| Random ums = new Random();  The seed value is milliseconds from the established origin point |

* How does C# generate a random number each time that my program calls the **nums.Next()** method?

|  |
| --- |
| The program goes through seed to generate the next nums.Next(). |

* If one creates a second Random object that is given the same seed value (as a previously created Random object) what will be true about the sequence of “random” numbers that it generates?

|  |
| --- |
| If the same seed is used, an identical set of numbers will be realized |

* Briefly give an example of where this might be useful:

|  |
| --- |
| Minecraft. Kidding. Seeing if they all test the same with the identical seed. |

* Given that the Random object is an adequate (but not great) random number generator, give an example or two of situations where you should avoid using the normal, **Random** class to generate random numbers:

|  |
| --- |
| Games; like e-gambling, e-sports, or lottery systems that drop items equivalent to real world currency. The last one is a personal vendetta. |

* Copy down the line that creates a Random object with a seed value of 100 from the example in the video

|  |
| --- |
| Random numTwo = new Random(100) |

**VIDEO: Random Numbers: the .Net-specific .Next(min, maxPlusOne) method**

* Pseudo-mathematically speaking, what do we mean when we say the numbers we’re generating are random?

|  |
| --- |
| Out of all the possible values the chances of being able to see all of them is equal |

* Given a Random object named nums, what possible numbers will **nums.Next()** generate?   
  Make sure to be clear about the smallest possible number, the largest possible number, and the type (double, float, int, etc)

|  |
| --- |
| A int between 0 -2.1billion |

* Given a Random object named nums, what possible numbers will **nums.Next(1, 7)** generate?   
  Make sure to be clear about the smallest possible number, the largest possible number, and the type (double, float, int, etc)

|  |
| --- |
| The lowest is 1, the largest is the next int under the limit which is 6. |

* Given a Random object named nums, what is the largest possible number that **nums.Next(1, 7)** might generate? (Yes, this duplicates what’s being asked for in the prior question, but getting this wrong is a very common mistake so it’s good to write your answer out twice)

|  |
| --- |
| 6 |

* Given a Random object named nums, what possible numbers will **nums.NextDouble()** generate?   
  Make sure to be clear about the smallest possible number, the largest possible number, and the type (double, float, int, etc)

|  |
| --- |
| NextDouble() can generate doubles at 0 and makes minute increments until the next int; which is 1. |

**VIDEO: Random Numbers: Seed Values & non-random numbers**

* Describe what will the following block of code produce as output (obviously you can’t know the exact numbers it will produce, so instead focus on what distinctive behavior it will demonstrate)

for (i = 0; i < 5; i++)

{

Random nums = new Random(); // seed value based on time

Console.WriteLine("Next number: {0}", nums.Next());

}

|  |
| --- |
| It will produce the same number since the Random keeps generating the same seed |

* **Why** does the code produce the output that it does in the prior question?

|  |
| --- |
| The same seed is being abused since its all happening in the same millisecond. After that its sent through the same ventures as its predecessors and future iterations. |

* How should you fix this problem?

|  |
| --- |
| Setting the new Random(); before the loop fixes it |

* Ideally, how many random number generators should your program create?

|  |
| --- |
| Ideally one, informally; for me, 6. One for each dice of Yahtzee. |

**VIDEO: Random Numbers: Using the plain .Next() & modulus ( % )**

* If I wanted to randomly generate whole numbers in the range of [1,5], what C# expression would I use?

|  |
| --- |
| (rng.Next() % 5) + 1; |

* Using the above expression, if **rng** randomly generates 1 what value will the overall expression produce? Make sure to explain each and every ‘step’ that the computer goes through in order to arrive at your answer.

|  |
| --- |
| Establish variables int 1  Test int 1 to % int 5  We get 1  Add int 1 to int 1  We get 2  2 is the final solution |

* What part of the expression (that you listed for the question that’s two before this question) determines how many values you get? Which part determines the lowest possible value that you’ll see?

|  |
| --- |
| The max value is based on the % sequence, the number added is to avoid ugly negative numbers so we can get the established lowest possible value. |