**Name: Patrick O’Brien**

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…"

1. 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)

|  |
| --- |
| The need to click on a onedrive link to view a video is extremely tedious. After watching the videos I felt annoyed so I looked up if you can embed them directly in the site. Turns out you can  <https://support.office.com/en-us/article/embed-files-directly-into-your-website-or-blog-ed07dd52-8bdb-431d-96a5-cbe8a80b7418>  This would increase the quality of the site dramatically if you could implement this for every video. That or use youtube, nobody downloads these to watch on desktop. Everyone however is familiar with youtube. |

**VIDEO: While Loops**

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

|  |
| --- |
| The while loops executes by running a loop while the Boolean logic is equal to true. When the while loop changes to false, the loop will stop. |

1. 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?)

|  |
| --- |
| What the user will enter in the while loop. The while loop might not know what the contents of the loop will be, and whether it will be true or false the first time. |

1. 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?

|  |
| --- |
| The user entering data, not sure when it will exit. It could loop until the user enters the correct parameter. |

1. 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)?

|  |
| --- |
| You don’t absolutely need the curly braces, without them it only runs the first line. It is recommended, but technically not required. |

**VIDEO: For Loops**

1. 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”

|  |
| --- |
| Made for counting: if you can know how many times you should execute the loop. |

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

|  |
| --- |
| For (int I = 0; I < 10; i++)  {    } |

1. What does **i++** do?

|  |
| --- |
| Increment the counter by 1 |

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

|  |
| --- |
| A semicolon |

1. 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?

|  |
| --- |
| No, you should initialize int = 0 once. |

1. 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?

|  |
| --- |
| Once |

1. 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?

|  |
| --- |
| Console.Writeline(“ I is: {0}, I); |

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

|  |
| --- |
| I++ |

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

|  |
| --- |
| Skips the increment and the body, loop ends, and it moves to the next part of the program outside the loop. |

**VIDEO: Do-While Loops**

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

|  |
| --- |
| A do-while loop executes at least once no matter the logic in the while statement. |

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

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

1. 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?

|  |
| --- |
| Many languages that are similar to C# ignore white space. An example of program that does not is python, since spacing is used instead of brackets. |

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

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

|  |
| --- |
| Decrements x by 1  Same as x -= 1  Stashes original value, increase memory block, then uses the new value. |

1. 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.

|  |
| --- |
| Int x = 10;  While (x > 0)  {  Console.WriteLine(“x is: {0}”, x--);  }  Prints 10 because it uses the original expression, but x will have the value 9. Prints current value then decreases it |

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

|  |
| --- |
| It works the same way as –x does except the decrement first and uses the resulting value in the expression. |

1. 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).

|  |
| --- |
| Int x = 10;  While (x > 0)  {  Console. WriteLine(“x is: {0}”, --x}  Console. WriteLine(“x is: {0} ===========”, x}  }  It prints 9 twice because with –x it ignores the original number and uses the newly calculated one for both statements. |

1. 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++**)

|  |
| --- |
|  |

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

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

|  |
| --- |
| Anything with C base/like has the operators, many languages |

1. 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 |

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

|  |
| --- |
| X += 2 |

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

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

**VIDEO: Random Numbers: Seed Values**

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

|  |
| --- |
| C# cannot generate a truly random number. Numbers are good at doing everything extremely precisely, and this extends to random numbers. |

1. 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 nums = new Random();  The seed value in the video will be 713 |

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

|  |
| --- |
| C$ will take 713 and grind it through a math function, generate a new number and hand it back to nums.Next  Then takes 212, and gets another one- 13  13 🡪 ???? |

1. 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?

|  |
| --- |
| The sequences of random numbers will be exactly the same. |

1. Briefly give an example of where this might be useful:

|  |
| --- |
| Using for tests, same sequence of tests to make sure the program is working correctly. |

1. 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:

|  |
| --- |
| Secure authentication programs, any kind of encryption. Gambling sites, any site with money. |

1. 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**

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

|  |
| --- |
| If given all possible values 🡪 no more likely to any particular value than you are to see another value. |

1. 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)

|  |
| --- |
| Generate a value including 0 to 2.16 billion |

1. 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)

|  |
| --- |
| A random integer including one, all the way up to but not including 7. |

1. 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)

|  |
| --- |
| A random integer including one, all the way up to but not including 7. |

1. 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)

|  |
| --- |
| A random double value specifically to 0 including 0 up to but not including the value of 1. |

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

1. 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());

}

|  |
| --- |
| Very distinctive 🡪 generate same number same time through the loop. So fast random stays the same after calling the clock. |

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

|  |
| --- |
| Because it happens so fast that the clock hasn’t changed millisecond by the time the loop is finished executing. |

1. How should you fix this problem?

|  |
| --- |
| Thread.sleep(milli), larger values, create random object outside the for loop. |

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

|  |
| --- |
| A single random number generator. |

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

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

|  |
| --- |
| (Rng.Next() % 5) + 1 |

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.

|  |
| --- |
| 1 % 5 = 1  + 1 = 2  2 = output |

1. 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 part of the expression that determines how many values = % 5  Part that determines lowest possible values = The + 1 |