Notes on 5.00 Practice Test (from 4/9/2019)

Here are some notes on the practice test.

Average Score: 79%

High Score: 100%

Low Score: 24%

|  |  |  |
| --- | --- | --- |
| **Ques #** | **Question Image** | **Notes** |
| 1 |  | If you missed this question, remember that a *Logic Error* gives an unexpected result; in other words, the program still runs. *Runtime* and *Syntax* errors will break and won’t run. *Dynamic Error* is a distractor (literally, the test writers are trying to trick/distract you…it’s not even a part of any of our curriculum). |
| 2 |  | SAME NOTE AS QUESTION #1: If you missed this question, remember that a *Logic Error* gives an unexpected result; in other words, the program still runs. *Runtime* and *Syntax* errors will break and won’t run. *Dynamic Error* is a distractor (literally, the test writers are trying to trick/distract you…it’s not even a part of any of our curriculum). |
| 3 |  | So, see the note on Question #1 and #2. However, to distinguish between Syntax and Run-time is the key here. With a syntax error, the program won’t compile (you can’t run it at all). With a run-time error, some sort of exception has been thrown like: an infinite loop, dividing by zero, or trying to convert a string (with letters) to a number data type (like integer, double, decimal). |
| 4 |  | This error would not compile as you would most likely be using this in the following context:Dim intNum As IntegerIf you misspelled “integer” the program wouldn’t compile and would be a syntax error as you wouldn’t be following the programming rules for this. |
| 5 |  | You all did a nice job with this one. If you missed it and don’t understand, please see me on Thursday or send me an email. |
| 6 |  | We haven’t really talked about this one much. You can probably infer based on what loops are as to what this is. We’re just tightening up our vocabulary here. |
| 7 |  | The key to this problem is to follow *intJ* and the comparison operator. By changing *intJ*  to subtracting 2 each time the loop runs, the condition at the end of the loop will never break…in other words, *intJ* will forever be less than 10 (in fact, heading in the wrong direction infinitely). -2, -4, -6, and so on forever.  |
| 8 |  | Great job on this one. FYI, intSum acts as an accumulator. In a *For…Next* loop, the variable after *For* is going to be your counter. Counters in *Do* loops look like:*intCount = intCount + 1* OR intCount += 1 This might have confused some of you looking for a counter that wasn’t part of the *For….Next* loop.dblAvg is a distractor: no part of the math involved with calculating it looks like a counter OR an accumulator. If you put this as an answer and need further explanation, please see me or email me so I can help you. |
| 9 |  | You need to approach this problem by doing the following table (track iterations along with the value):

|  |  |
| --- | --- |
| Iteration | intJ |
| Initialized | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |

The loop executes 5 times since the value going into the 5th loop is 8 (which is less than 10). It won’t execute a 6th time since 10 is *equal* to 10 (not less than). |
| 10 |  | Pretty straightforward question and easy to fix. Right? |
| 11 |  | OK, you should know what I’m about to recommend. DO A TABLE. Remember, intCount is stepping by 2 at a time and starting at 0. intResult is a classic counter, incrementing by 1 each time. Here’s the table:

|  |  |
| --- | --- |
| intCount | intResult |
| 0 | 1 |
| 2 | 2 |
| 4 | 3 |
| 6 | 4 |
| 8 | 5 |
| 10 | 6 |

The next loop would result in intCount being 12 and therefore would not run since the max value in the range is 11. Be this deliberate when you’re trying to figure out these problems. |
| 12 |  | So, I understand why some of you answered the loop wouldn’t start. The state didn’t give you the whole *For…Next* loop structure. If you see a question like this, understand that they want you to assume the rest of the code is there. I know the frustration here and I agree with you 😉. |
| 13 |  | Yikes! We didn’t do well on this one. Let’s do a TABLE (revolutionary, I know). We know the loop will stop when *intCount* is greater than 3, so let’s track it and *intNumber*.

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | intCount | intNumber | Notes |
| Initialized | 0 | 0 | Initialized values before the loop starts |
| 1 | 1 | 2 |  |
| 2 | 2 | 4 |  |
| 3 | 3 | 6 | Here is the key: intCount has to be greater than 3 it is equal to three at this point, so the loop runs again and THEN exits, but updating the values before it does so. |
| 4 | 4 | 8 | 4 is greater than 3, so the loop exits at this point. |

 |
| 14 |  | So, you’re really on tracking *intJ* here. It’s going to run as long as *intJ* is less than 10 (remember that this is a posttest loop). Here’s the table tracking iterations and values:

|  |  |
| --- | --- |
| Iteration | intJ |
| Initialized | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |

Again, we encounter a situation where the loop has run and exits at the END…it won’t run a 6th time since 10 = 10 (in other words, 10 isn’t less than 10, which is the condition to exit the loop). Because it is a posttest loop, the values will update BEFORE exiting the loop that 5th time. |
| 15 |  | OK, you just have to go with basic definitions here. An *accumulator* updates by a *varied* amount (as opposed to a *counter* which updates by a *set* amount). We know that *intSum* is updating by a varied amount each time since *intGrade* generates a random number each time.If you gave *intCount* as an answer realize that it’s position in the *For…Next* loop makes it a counter (this is the syntax for this type of loop as it will increment by 1 each time…a set amount).An accumulator doesn’t look like *dblAvg* (the math on an accumulator is simple: repeated addition or subtraction most of the time).If you put down *intGrade*, you need to realize that it is assigned a random number each time and no accumulated (running) total is given for it between loops (it resets each time); therefore, it can’t be an accumulator (nor can it be a counter). |