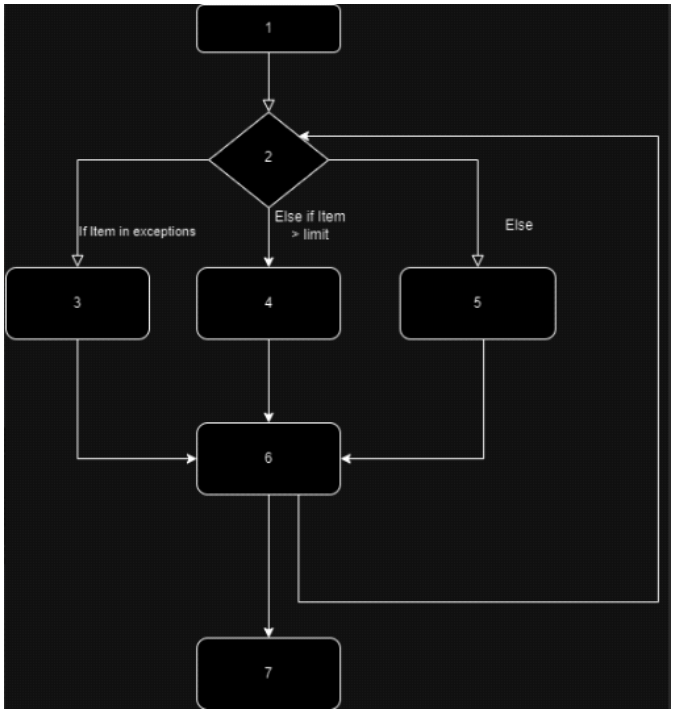
Assignment 1

October 15, 2023 3:16 PM

1. A "sound" test is one that always accurately finds problems but not necessarily *all* the problems, while a "complete" test will find all the problems but not necessarily *only* the problems.

   True Positives: Correctly identifies any found problems as such
   False Positives: Incorrectly identifies non-issues are problems
   True Negatives: Correctly identifies non-issues as such
   False Negatives: Incorrectly identifies problems as non-issues

   These terms technically wouldn't need to change even in the event that the goal means that "positive" means *not* finding a bug as in this case, finding a bug is simply considered a non-issue/negative. However, for the sake of clarity it would be easier to simply flip the usage of the terms.
2.
   b. "*Run until an array of each size has been created at least once*"
3.
   a. 



b. You could use inputs of random items that fit the different criteria for the paths and check to see what the outcomes end up as.

4.

a.

i. A group of items with at least 1 in exceptions, 1 above the limit, and 1 that is neither (100% code coverage)
ii. A group of items with at least 1 in exceptions and 1 that's neither in exceptions nor above the limit, but none that are above the limit (93% code coverage)
iii. A group of only items that're neither in exceptions or above the limit (71% code coverage)
iv. A group of only items that're either above the limit or neither above the limit nor in exceptions (93% code coverage)

5.

a. The main problem with this code is in the "elif char.isnumeric():" statement as since the code snippet is supposed to leave numeric characters unchanged, multiplying the char's value by 2 would of course, change the character. Otherwise, if the code was *meant* to multiply the numeric character by 2, it would still produce incorrect results as this would multiply the character's *ASCII* value instead of the numeric value, producing incorrect results (e.g. '1' * 2 would become 98 since the ASCII value of '1' is 49)
b. My delta-debugging algorithm uses a binary search as its base, it checks the entire list of inputs for any bugs by denoting what the correct outcome should be and looking to see if they're equal, then if a bug is found it either checks the 1st set (if the 1st or both sets have bugs) or the 2nd set (if only the 2nd set has bugs) and runs through it again to see which case has the problem.