Project 1. You are to write (and test) software that rolls 6-sided dice.

Details. The software should be written as a potentially reusable component for other programs, such as games or simulations. Because not all games require two dice to be rolled (for example, some games, such as Trivial Pursuit, use a single die; while others, such as Risk, require three), the component should be able to "simultaneously" roll up to five dice at a time.

The user should be able to specify how many dice need to be rolled, between 1 and 5. The software should then return that many rolls of dice, independently generated.

Example output (the printing part of this should be done separately from the dice-rolling routine itself; the dice-rolling routine should only return these values as output parameters to the calling routine):

***Single die (5 runs)*:**	***Two dice (5 runs)*:**	***Four dice (5 runs)***
3	5 6	3 3 1 1
1	4 5	4 6 2 2
6	3 1	5 6 5 1
4	5 5	3 4 1 6
3	2 1	4 2 3 2

Testing. You should test your software, making sure it behaves as expected, to include (here's the tricky part) making sure it sufficiently emulates randomness. Think about what this means, and be sure to construct test cases that test this property of the output.

Documentation. Your project submission should include your source code, output of sample runs, and your test plan. Also document your assumptions.

Remember, the test plan consists of several test cases, and their order of execution. Each test case has these components, as discussed in class: Inputs, Expected Outputs, Actual Output, and Test Case Result. Be sure to indicate whether or not the software has passed each test – this may seem like a redundant step, but how can a reviewer know for sure that a test has indeed been run, unless such annotations are made?

Note: If your testing discovers an error in your code, do not merely fix the error, and then rerun the test case. Document the error first, then fix the code. Your final turn-in should contain, in the Actual Results section, BOTH the results of the failed test, along with the results of the subsequent test. A brief note explaining what was done to correct the problem is considered a good practice, and should be employed throughout this course.

Remember, the bulk of your grade will depend upon the documentation – most especially the test plan – and not the code itself.

As discussed in class, I'd expect to see the following tests (at a minimun):

- Distribution tests for a single die.
- Distribution tests for multiple dice.
- Tests demonstrating independence from previous results.

You could (should?) also do some checking for things like testing for doubles with two dice. How often is this expected to occur? How often does it occur over a large sample size?

Error checking is NOT the focus of this assignment. It's best to simply omit such test cases from your test plan. (You can assume another department is responsible for error checking.) The bulk of your grade in this assignment will be based on the **rigor of your randomness testing**. Additional randomness checks (aside from the tests mentioned above) are encouraged.

As discussed in class, some level of automation will need to be employed in order for this to be accomplished. The dice program will be trivial but writing the test harness will NOT be a trivial task.

Here are some additional tips for getting a good grade:

1. Include a short introduction, which would include instructions for Alice as well as any assumptions you have made that should be shared.

2. There should be cases that test for normal distributions, using both single and multiple dice. There should also be "dependency tests," showing that dice rolls seem unaffected by previous rolls.

3. Make your test plan easy to use, easy to review, and easy to follow. For example, when using tables to set up your test cases, avoid putting Expected Results and Actual Results in two separate tables. Not only does this take up more room, this format makes the results harder to check. Oftentimes, combining the expected and actual results into a single table is easier to analyze than using two separate tables. Test this yourself: Which of the following formats is easier to check?

Two-table format:

Count	Units	Exp. Range
RBC	c/mcL	4.32 to 5.72
WBC	c/mcL	3500 to 10500
Platelets	mcL/1000	150.0 to 450.0
Homoglob.	g/dL	13.5 to 17.5
Hemat.	pct	38.8 to 50.0
LDL	mg/dL	40 to 60
HDL	mg/dL	100 to 160

Count	Units	Actual Val.
RBC	c/mcL	4.38
WBC	c/mcL	7502
Platelets	mcL/1000	300.1
Homoglob.	g/dL	18.8
Hemat.	pct	37.7
LDL	mg/dL	56
HDL	mg/dL	151

Single-table format:

Count	Units	Exp. Range	Actual Val.
RBC	c/mcL	4.32 to 5.72	4.38
WBC	c/mcL	3500 to 10500	7502
Platelets	mcL/1000	150.0 to 450.0	300.1
Homoglob.	g/dL	13.5 to 17.5	18.8
Hemat.	pct	38.8 to 50.0	37.7
LDL	mg/dL	40 to 60	56
HDL	mg/dL	100 to 160	151

4. Where possible, use page breaks to avoid having tables be spread across multiple pages. It is better to start each test case on a new page than to have tables being broken across multiple pages. (That said, it's okay to put two or more test cases on a single page, so long as the last one doesn't spill onto the next page.)

5. Your final project should include your completed test plan, as well as your code. The code includes the code for your dice program as well as for your testing harness. The code should be after the test plan, and any help you got from AI sources should be listed after that.