# Overview

This brief coding exercise will help us better understand how you think through problems and design solutions. Most importantly, remember that we’re looking for solutions that are concise, meet all the stated and implied requirements, exhibit clarity of design and intent, maintain simplicity while being thorough, and make deliberate tradeoffs in compute time and memory management.

# Background: Classic FizzBuzz

Classic fizzbuzz output is typically described as “Write a program that prints the numbers from 1 to 100. But for multiples of 3, print “Fizz” instead of the number. For multiples of 5, print “Buzz”. For Multiples of 3 and 5, print “FizzBuzz”. Typical fizzbuzz output looks like:

1  
2  
Fizz  
4  
Buzz  
…  
88  
89  
FizzBuzz  
91  
….

…and so on. But that’s way too simple!

# Requirements: Super FizzBuzz

1. Develop a class library called SuperFizzBuzz that can do the following:

* Can produce fizzbuzz output for a user-supplied range of numbers – e.g., from 2 to 35, or from 1 to 1,000,000,000, or from -1 to -35, etc.
* Can produce output for a user supplied set of integers, even if they’re not sequential.
* Can generate tokens other than “Fizz” and “Buzz” and can evaluate division by numbers other than 3 and 5. Maybe a user wants to test division by 4, 13, and 9, and output “Frog”, “Duck,” and “Chicken” for them (e.g., in this case, 52 would ouput “FrogDuck”, 36 would output “FrogChicken”, 468 would output “FrogDuckChicken”, etc.)

1. Write a console application, using your SuperFizzBuzz class, to solve the classic FizzBuzz problem as described above in “Classic FizzBuzz”.
2. Write a second console application demonstrating advanced usage of SuperFizzBuzz that performs the following:
   1. Print the numbers from -12 through 145.
   2. For multiples of 3, print “Fizz”
   3. For Multiples of 7, print “Buzz”
   4. For Multiples of 38, print “Bazz”
   5. Print the appropriate combination of tokens for any number that matches more than one of those rules
3. In your solution, include tests for your SuperFizzBuzz class and any other related classes you develop.
4. Put all your work in a zip file and upload it with the results from your questionnaire to the link we provided you.

# Common Pitfalls to Avoid

* Your final solution should include at least four projects – one for your class library, two for the console applications, and at one or more projects for automated tests.
* Your console applications do not need to parse any command line arguments. They just need to generate output, using your SuperFizzBuzz class, as described in requirements 2 and 3.
* Even though we’re the authors of AutoMapper, MediatR, and lots of other open-source libraries, there’s no need to import or use any of them for this exercise. We’re not looking for solutions that show off every possible language feature, that are over-engineered.
* It’s important to keep it only as complicated as it needs to be, but conversely we do value quality and it is important the submission does not feel rushed or unpolished