# **Code Quality Assurance Practice**

## First

* Install Node.js:
  + <https://nodejs.org/>
* Install Mocha (a JS TDD framework):
  + <https://mochajs.org/#installation>
* Navigate to the template folder and run: *mocha test.js*

## We Are Practicing

* Test driven development (TDD) process
  + **Red**: Understand the (user) requirements well and write a (initially failing) test for what you expect.
  + **Green**: Write (only) the code you need to make the (failing) test pass, while ensuring your existing/previous tests all still pass (no regressions).
  + **Refactor** the code you wrote - tidy up the code you wrote to make it easier to understand and maintain.
* Pair programing
  + One programmer “drives,” operating the keyboard, while the other “navigates,” watching, learning, asking, talking, and making suggestions.
  + Make sure you **talk a lot.**
  + Switch roles after one hour of coding.
* Incremental development
  + Focus on one small, self-contained “increment” at a time.
  + Each successive increment should always make the product readily usable.
* Git
  + add, commit, push, etc.
  + Make sure to commit in small, self-contained “increments”.

## Code Quality Requirements and Notes

* Consider each item in functional requirements as an iteration (i.e., increment). In each iteration, write the test code first, then write code to pass the tests.
* Commit your code to Git after each iteration (i.e., each passing test).
* Remember the golden rules for code quality:
  + Write in good, consistent style; use expressive names
  + Write small, single-responsibility functions and classes
  + Do not repeat code
  + Solve things as simply as possible
  + and so on…
* Remember to refactor (if needed) after each passing test.

## Functional Requirements

1. Create a simple String calculator with a method *int add(numbersString)*
   * For an empty *numbersString* it will return 0
     + i.e. *add(“”)* returns 0.
   * For a single number it will return its value.
     + e.g. *add(“2”)* returns 2; *add(“14”)* returns 14; etc.
2. Allow the Add method to take a String of up to 2 numbers (separated by ‘,’), and will return their sum
   * e.g. *add(“2,3”)* returns 5; *add(“12,5”)* returns 17; etc.
3. Allow the Add method to handle an arbitrary amount of numbers
   * e.g. *add(“2,3,5”)* returns 10; *add(“11,10,2,3,5,1,3”)* returns 35; etc.
4. Allow the Add method to handle new lines between numbers (instead of commas).
   * the following input is ok: “1\n2,3” (will equal 6)
   * the following input is NOT ok: “1,\n”
5. Support different delimiters
   * to change a delimiter, the beginning of the string will contain a separate line that looks like this:“//[delimiter]\n[numbers…]”
   * for example “//;\n1;2” should return three where the default delimiter is ‘;’ .
   * the first line is optional. All existing scenarios should still be supported.