

Professional Practice Assignment 1 --- Intro to Unit Testing & T/BDD

(teams of up to 2 allowed for this assignment)

Objective

Apply { test | behavior }-driven development to implement simple functions. Write unit tests to provide adequate coverage using your chosen programming language, unit testing framework, test runner, and required tooling. Document functionality with tests (naming and / or test descriptions). Assignment can be done **individually *or* with a partner** (both students will receive the same grade).

Requirements:

Command Line Interface - Develop a command line app that prompts the user to select a function to execute, provide input, get a correct response, select another function *or* allow the user to gracefully exit the app. The menu should be displayed after each function unless the user exits. Apply Test / Behavior-Driven Development to implement the command line app. The unit test coverage for the application should be greater than ~85% (can skip CLI presentational code). READ: "Try this BDD Example" <http://bit.ly/2k4Fdj2>

Choose any ***4*** of the 5 functions below to test and implement. Stated requirements must be met for 'correct' function.


1. **Body Mass Index** - Input height in feet and inches. Input weight in pounds. Return BMI value and category: Underweight = <18.5; Normal weight 🍌 = 18.5–24.9; Overweight = 25–29.9; Obese = BMI of 30 or greater (need to convert height and weight to metric values - see formula linked below)
2. **Retirement** - Input user's current age, annual salary 💰, percentage saved (employer matches 35% of savings). Input desired retirement savings goal. Output what age savings goal will be met. You can assume death at 100 years (therefore, indicate if the savings goal is not met 💀).
3. **Shortest Distance** - Input 2 points (x1, y1) (x2, y2) [4 values - indicate to user which values are being input] and calculate the distance between the points using the distance formula (*should be implemented without use of libraries with the exception of a **square root** library function*). Floating point precision determination is key for testing.
Distance Formula: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
4. **Email Verifier** - Input a string, determine if it is a valid email address (i.e., some_string '@' domain) according to these stated requirements with required caveats. some_string1 can consist of text, optionally separated by periods. No periods can start or end some_string1. Two periods together is invalid and the address must start with a non-numeric character. some_string1 can contain the following printable characters: `!$%*+,-=?^_{}~` but not: `"()<,>@[\\`` (this function provides a good opportunity to use regular expressions).
5. **Split the Tip** - Input a total dinner amount (ex. \$55.40) and the number of guest to split the tab with. Add 15% gratuity to the total. The function should calculate the total tab (including tip) and as equally as possible distribute the cost of the entire meal between the number of provided guests. (e.g., splitTip (\$15.16, 3) ==> guest1-\$5.06, guest2-\$5.05, guest3-\$5.05). Whatever logic is used for uneven amounts should be deterministic and testable (all values rounded to cents - 2 decimal places).

Assignment

Project Report (Posted to Github Repo as Readme markdown file): Write a report summarizing your efforts. Your report should consist of the following content:

- **Discuss naming and organizational conventions:** Determine (based on the framework / language selected) how you will name and organize your unit tests. Briefly describe (once for all tests).
- **Detailed setup and execution instructions (put in README on main page):** include links to download (languages, environments), instructions to setup the test framework and execute the application (and tests). I should be able to execute your instructions on a new Ubuntu or fresh Windows 10 install. Be explicit. State which versions of tools used (e.g., python 2.7.x) and what environment/OS to run on. Assume that the grader doesn't have any experience with the language you chose or the testing frameworks etc.
- **Screenshot or report output showing all tests suites & tests passing:** This output should provide documentation for each function.
- **Test coverage report** - Report showing code coverage for the entire application. You will probably need a third party tool to evaluate and report coverage. Search for an appropriate tool for your chosen framework / programming language. Show coverage details for each function. ***Research how to generate coverage reports for your chosen language / framework***
- **For each student (i.e., individually if working with a partner - write name next to answer):** Describe your unit testing & TDD experience. What do you think of unit testing & TDD? Do you think it's useful for a real project? Benefits & Drawbacks to TDD.

Screencast - Evidence of Test First Process - Red, Green, Refactor

- Suggested to use LiceCap (<http://www.cockos.com/licecap/>) - OBS Studio is also another open source option (<https://obsproject.com>)
- **Record** a screencasts of 2 iterations of the red, green, refactor process for separate functions (no sound required)
 - Prior to recording, complete initial test fixture setup and several unit tests with associated implementations (i.e., you should have completed several RGR iterations with some functional code previously implemented - so your recording should not be your first test).
 - Record up to 2 minutes of desktop (IDE / Editor / Terminal) + test runner output that shows at least 2 RGR steps:
 - 1. Write a test - watch it fail
 - 2. Write enough code so that the test passes
 - 3. Refactor (optional - as needed)
 - Repeat 
 - Objective is to show process conformance
- **Record** execution of ***ALL*** functions in the working program (2-3 minute max video)
 - Show the program's initial screen, select a function, provide input, execute, and display results (repeat * 4 for all functions)
- Upload 2 screencasts and all related code and data to GitHub Repo

Notes & Resources:

- BMI Formula - <http://extoxnet.orst.edu/faqs/dietcancer/web2/twohowto.html>
- Distance Formula - <http://www.purplemath.com/modules/distform.htm>
- **Remember**, unit tests serve as documentation - your tests should indicate the features of the program
- Be sure that each function and test follow the same organizational and naming conventions
- See in-class demo code (8/30) here: <https://github.com/drbyron-github-classroom/titlecase>

Your report (including screencast) should be contained in your GitHub repo. The wiki will be used to prepare the report. Link any images (screenshots) or screencasts in-line back to the raw file stored in the Github Repo. File README should contain setup and execution instructions and are not required to be included in wiki report.

SUBMIT

Link to Github Repo - No Changes accepted after Due Date

Due Tuesday, September 10 **11:59pm** (late submissions penalized)

Professional Practice Points: **70**

100%