Karen Shay West
9 Shannon Marie Way, North Easton, MA 02356

508-844-9776 http://www.linkedin.com/in/karenshaywest KarenWest15@gmail.com

SUMMARY OF RECENT ONLINE TECHNICAL in Git, GitHub and Ruby On Rails for Web Apps Introduction – see my Linked In Project Section for details and github links to files:

- **Git and GitHub:** Effective use of version control is an important and useful skill for any developer working on long-lived (or even medium-lived) projects, especially if more than one developer is involved. This course, built with input from GitHub, covers the basics of using version control by focusing on a particular version control system called Git and a collaboration platform called GitHub.
- **Git** is used by many tech companies, and **a public GitHub profile** serves as a great **portfolio** for any developer. You establish an **efficient programming workflow** that allows you to:
  - Keep track of multiple versions of a file
  - Track bugs by reverting to previous working versions of a file
  - Seamlessly collaborate with other developers on a project
  - The use of tools like Git and GitHub is essential for collaborating with other developers

- **Topics:**
  - **1: Navigating a Commit History:** We learned about a few different types of version control systems and discovered what makes Git a great version control system for programmers. We also got practice using Git to view the history of an existing project. We saw all the versions that had been saved, checked out a previous version, and compared two different versions.
  - **2: Creating and Modifying a Repository:** We learned how to create a repository and save versions of our projects. We learned about the staging area, committing our code, branching, and merging.
  - **3: Using GitHub to Collaborate:** We practiced using GitHub or other remote repositories to share our changes with others and collaborate on multi-developer projects. We learned how to make and review a pull request on GitHub. At the end, we practiced by collaborating with other students to write a create-your-own-adventure story.
  - **Project: Contributed to a Live Project:** We published a repository containing our reflections from the course and submitted a pull request to a collaborative Create-Your-Own-Adventure story.

- **Ruby on Rails Software as a Service Web Applications (Introduction):** Fundamentals of software engineering using Agile techniques to develop Software as a Service using Ruby on Rails. Taught the fundamentals for engineering long-lasting software using highly-productive Agile techniques to develop Software as a Service (SaaS) using Ruby on Rails. Learned to understand the new challenges and opportunities of SaaS versus shrink-wrapped software. Learned to understand and apply fundamental programming techniques to the design, development, testing, and public cloud deployment of a simple SaaS application. We used best-of-breed tools that supported modern development techniques including behavior-driven design, user stories, test-driven development, velocity, and pair programming. We learned how modern programming language features like metaprogramming and reflection can improve productivity and code maintainability. REST methods for applications was covered, along with HTML and

CSS formatting of web page and database entries.

- **Ruby Calisthenics (exercises to learn this scripting language):**
  - Problem 1 covered **Palindromes and Word Count functions** in Ruby.
  - Problem 2 covered a **Rock-Paper-Scissors game.**
  - Problem 3 covered **anagrams**.
  - Problem 4 covered **basic object-oriented programming (OOP)**. We made a dessert class with getters and setters for name and calories. We also defined the healthy? and delicious? methods. We made a class JellyBean that inherited from Dessert class, but redefined the delicious? method of Dessert to say that black licorice was not delicious.
  - Problem 5 covered **advanced OOP, metaprogramming, open classes, and duck typing**. We created a method attr_accessor_with_history method that tracks every value the attribute has ever seen.
  - Problem 6 also covered **advanced OOP, metaprogramming, open classes, and duck typing.** It was a currency converter. We also rewrote the palindrome ruby function done in problem one as a method within the foo class. The last part adapted the palindrome method to work with enumerables.
  - Problem 7 dealt with **Ruby iterators, blocks and yield by having us write a Cartesion Product example.** The constructor accepts 2 list sequences. We defined the each instance method of CartesianProduct, which yields the cartesion product of the 2 sequences used in the class constructor. The iterator each should yield the values one at a time as 2 elements arrays.
- **Introduction to Rails, the Ruby Framework for Web Applications:**
  - We added some movies to the database, and deployed the RottonPotatoes app that displays the list of movies in a web browser publicly on Heroku. On the RottenPotatoes “All Movies” page, the column headings for “Movie Title” and “Release Date” for a movie had to be clickable links. Clicking one of them should cause the list to be reloaded but sorted in ascending order on that column. When the listing page is redisplayed with sorting-on-a-column enabled, the column header that was selected for sorting should appear with a yellow background. We did this by setting controller variables that are used to conditionally set the CSS class of the appropriate table heading to “hilite”, and pasting a simple CSS snippet given to us into RottenPotatoe's app/assets/stylesheets/application.css file. The current RottenPotatoes views use the Rails-provided “resourceful routes” helper “movies_path” to generate the correct URI for the movies index page. Passing this helper method a hash of additional parameters, those parameters will be parsed by Rails and become available in the params[] hash. Next, functionality added within RottenPotatoes that allowed the user to filter movies by MPAA rating such as ['G', 'PG', 'PG-13', 'R']. The filter was controlled by check boxes at the top of “All Movies” listing. When the “Refresh” button was pressed, the list of movies is redisplayed showing only movies whose ratings were checked. Next we remembered the settings - if the user selected any combination of column sorting and restrict-by-rating constraints, and then the user clicks to see the details of one of the movies, when she clicks “Back to the movie list” on detail page, movie listing should “remember” the user's sorting / filtering settings from before. “remembering” - use the session[] hash.
  - **Behavior Driven Design:** We created user stories to describe a feature of Saas application using the Cucumber tool to turn user stories into executable tests. The tests ran against the Saas app. We wrote Cucumber scenarios that test the happy paths of the previous Introduction to Rails assignment described above. We used declarative

scenario steps created for adding movies. The goal of Behavior-Driven Design (BDD) is to express behavioral tasks rather than low-level operations. The background step of the scenarios in this homework required that the movies database contain movies. It is not the goal of BDD to do this by writing scenarios that spell out every interaction required to add a movie, since adding new movies is not the scenario test goal. The given steps of a user story specify the initial state of the system. It does not matter how the system got into that state. For Part1 of this assignment, we created a step definition that matches the step "Given the following movies exist" in the background section of sort_movie_list.feature_and_filter_movie_list.feature. We then added our code to the movie_steps.rb step definition file. In the Active Record, we added movies to the database, since creating new movies is not what the scenario is testing. Success was defined as when all Background steps for the scenarios in filter_movie_list.feature and movie_list.feature were passing (green). For Part 2 of this assignment, we added Happy Paths to filter movies, restrict the movies with 'PG' and 'R' ratings in the filter_movie_list.feature. It allowed us to use existing steps in the web_steps.rb to check and uncheck the appropriate boxes, submit the form, and check whether the correct movies appear, ignoring the case where the movie has no ratings. Part 3 of this assignment was to add Happy Paths by sorting movies by title and release date. Scenarios in sort_movie_list.feature involve sorting, so we needed the ability to have steps that test whether one movie appears before another in the output listing. We had to create a step definition that matched a step such as that.

    - **The last topic covered in this course was Test Driven Development (TDD).**

- **Topics covered in Part 2 of this course (not taken as of yet) include:** creating more sophisticated apps by adding relationships between models in apps and by enhancing their apps with JavaScript. You also learn about what happens after the apps are deployed to real users, including how to monitor performance, identify and fix common performance problems, and avoid compromising customer data. You learn how to apply Agile techniques to enhance and refactor legacy code, a critical skill for professional programmers. Other topics covered in part 2 of this course include: How to form, organize and manage small programming teams, Introduction to design patterns: what they are and how to recognize opportunities to apply them, Using Rails for more advanced features like third-party authentication and elegantly expressing design patterns that arise frequently in SaaS.