🚋

Setting up a new Project TO DOs

Gemfile

The very first step of starting a Ruby project is to initialise it with a Gemfile. Gemfiles act as a reference for all the libraries (‘gems’) we’re going to use in this project. We will use Bundler, a dependency manager, to work with the gems listed in the Gemfile.

Setting up the project

1. Install RSpec

- [ ] gem install rspec

2. Setting up a new repository in your projects directory adding a README.md file

- [ ] mkdir oystercard
- [ ] cd oystercard
- [ ] git init
- [ ] touch README.md

3. Add a basic explanation of the project to the README.md and then commit changes

- [ ] git add README.md
- [ ] git commit -m "first commit"
- [ ] git branch -M main

4. Make a remote repo on GitHub (do not initialise it) and name it the same name as the local repo

- [ ] git remote add origin <github link.git>
- [ ] git push -u origin main

Creating RSpec conventional files

5. Running the command line option `-init` in your project directory will create a `spec/` folder with `spec_helper.rb` inside it, and a `.rspec` file in your project directory.

- [ ] rspec --init

6. Create an empty test suite for the oystercard class

- [ ] `touch spec/oystercard_spec.rb`

7. Git add, commit and push all the spec files

- [ ] git add .
- [ ] git commit -m "add rspec conventional files"
- [ ] git push origin main

Debugging basics

A stack trace shows a list of method calls that lead to the exception being thrown, together with the filenames and line numbers where the calls happened.

Pull request vs forking the repo

A pull request is simply a request to merge (or “pull in”) changes from your repository to theirs repository. **Forking** is making a copy of a repository from someone else’s github to yours, which you then own. You can make changes to your forked repo,

and do whatever you like with it. You can then make a PR to request that the original repository is updated to include your changes.

RSpec Syntax

Built in matchers

rspec-expectations ships with a number of built-in matchers. Each matcher can be used with expect(..).to or expect(..).not_to to define positive and negative expectations respectively on an object. Most matchers can also be accessed using the(...).should and (...).should_not syntax; see using should syntax for why we

https://relishapp.com/rspec/rspec-expectations/docs/built-in-matchers

Require vs Require_relative

Require is a method that is used when you want to reference and execute code that is not written in your current file.

The method takes in a path in the form of a string as an argument and there are two ways the string can be formatted — either as an absolute path or a shortened name.

```
# absolute path
require './app/example_file.rb'

# shortened name
require 'example_file'
```

Require_relative vs Require_all

Though `require` can be used to both execute gems and external dependencies, the preferable method to load relative paths is `require_relative`. `require_relative` is a subset of `require` and is a convenient method to use when you are referring to a file that is relative to the current file you are working on (basically, within the same project directory).

The general rule of thumb is `require` should be used for external files, like gems, while `require_relative` should be used for referring to files within your directory.

Though you *can* call on absolute paths using both methods, but `require_relative`'s scope is wider and is aware of the entire directory where the program resides

🧩 Interview questions

🌈 RSpec::Expectations Cheat Sheet Ruby

🏳️‍🌈 RSpec::Core Cheat Sheet