Week 12 - Testing, Tools and Projects

Overview

- Portfolio Submission
- Tools:
  - Testing
  - Projects
  - Debugging
  - Profiling

Portfolio Submission

Do not forget to complete your Learning Summary (TT 11.1).

This is counted as completing your portfolio - in it you identify the tasks we will consider for marking your portfolio and we also consider your reflection when determining your grade.

Your final grade will depend on everything you have submitted.

Tools – Testing I

Do both positive and negative tests:

```
require 'minitest/autorun'

class TestMyThing < Minitest::Test

  def test_that_it_works
    # ...
  end

  def test_it_doesnt_do_the_wrong_thing
    # ...
  end

  # ...
end
```

Source: Fulton, H & Arko, A 2015 The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Tools – Testing II

You can use create setup and teardown methods for the class. These methods are called before and after every test:

```
def setup
  # ...
end

def teardown
  # ...
end

def test_it_works
  # ...
end

def test_it_is_not_broken
  # ...
end
```

Source: Fulton, H & Arko, A 2015 The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Tools – Testing III

You can use assertions:

```
assert_equal(expected, actual) # assert(expected == actual)
refute_equal(expected, actual) # assert(expected != actual)
assert_match(regex, string)    # assert(regex =~ string)
refute_match(regex, string)    # assert(regex !~ string)
assert_nil(object)             # assert(object.nil?)
refute_nil(object)             # assert(!object.nil?)

assert_instance_of(klass, obj) # assert(obj.instance_of? klass)
assert_kind_of(klass, obj)     # assert(obj.kind_of? klass)
assert_respond_to(obj, meth)   # assert(obj.respond_to? meth)
```

Source: Fulton, H & Arko, A 2015 The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional.

Tools – Testing IV

The code `quadratic_test.rb` here tests a program called `quadratic.rb`.

- Source: Fulton, H & Arko, A 2015 The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Digression - Exception Handling

Try out the following commands to see the `exception.rb` code run:

```
MacBook-Pro-8:Testing mmitchell$ ruby ExceptionHandling.rb

Enter filename:
red
failed to open red
Please reenter

Enter filename:
sdd
failed to open sdd
Please reenter

Enter filename:
ExceptionTest.rb

 Successfully Opened ExceptionTest.rb
MacBook-Pro-8:Testing mmitchell$
```

Tools - Projects

- RubyGems
- Bundler
- Rake

Tools - Ruby Gems I

Rubygems.org is the community-funded host for public gems. This site creates a page for every gem that lists the gem's authors and versions.

Installing gems

–Use the gem tool.

–Eg: to install the rake gem, you would run `gem install rake`. Rubygems will look for a .gem file in the current directory, and then check rubygems.org.

–You can install earlier versions: eg: to install rake version 10.0.1, you could run `gem install rake -v 10.0.1`.

> Source: Fulton, H & Arko, A 2015, The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Tools - Ruby Gems II

ruby-toolbox.com provides a list of gems and what they do.

Use `gem list` to see what gems you have installed.

Uninstall gems using `gem uninstall NAME`.

> Source: Fulton, H & Arko, A 2015, The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Creating Gems I

Create a file with `.gemspec` extension

Eg: a sample `.gemspec` file for a gem named drummer:

```
# drummer.gemspec
Gem::Specification.new do |spec|
  spec.name          = "Drummer"
  spec.version       = "1.0.2"
  spec.authors       = ["H. Thoreau"]
  spec.email         = ["cabin@waldenpond.net"]
  spec.description   = %q{A Ruby library for those who march to a differ
  spec.summary       = %q{Drum different}
  spec.homepage      = "http://waldenpond.com/drummer"
  spec.license       = "MIT"

  spec.files         = Dir["./**/*"]
  spec.executables   = Dir["./bin/*"]
  spec.test_files    = Dir["./spec/**/*"]
  spec.require_paths = ["lib"]

  spec.add_development_dependency "rake"
  spec.add_runtime_dependency "activerecord", "-> 4.1.0"
end
```

Source: Fulton, H & Arko, A 2015, The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Creating Gems II

Build a gem by running the command:

```
gem build drummer.gemspec
```

Then, to upload the gem to rubygems.org run:

`gem push drummer-1.0.2.gem`

Refer to the Rubygems guides at http://guides.rubygems.org

> Source: Fulton, H & Arko, A 2015, The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Bundler I

How to manage dependencies?

If you have a large application that uses many gems, and you might have different applications that use different versions of gems.

Create a gemfile and list the needed gems inside:

```
source "https://rubygems.org" # Download gems from rubygems.org

gem 'red'                   # A dependency on a gem called "red"
gem 'green',  '1.2.1'       # Gem "green" - version 1.2.1 exactly
gem 'blue',   '>= 1.0'      # Version 1.0 or greater of gem "blue"
gem 'yellow', '~> 1.1'      # "yellow" 1.1 or greater, less than 2.0
gem 'purple', '~> 1.1.1'    # At least 1.1.1, but less than 1.2
```

Source: Fulton, H & Arko, A 2015, The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Bundler II

Run bundle install to install all the required gems.

Creates a Gemfile.lock

the lock stores the names and exact installed versions of every required gem, including the dependencies of your gems, the dependencies of those dependencies, and so on.

To run a gem command using the correct version for a given project, prepend the gem command with bundle exec.

Eg: Instead of running `rake spec`, you would run `bundle exec rake spec`.

This guarantees that the version of rake that is used will be the version that the project requires.

See http://bundler.io for more info.

> Source: Fulton, H & Arko, A 2015, The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Rake I

The Rake utility is like a Ruby version of the UNIX make utility

Allows you to define tasks, declare dependencies between tasks, and then run a task and all of its dependencies at once.

Uses a file of instructions called the Rakefile.

The basic “unit of work” in Rake is the task; these are named with Ruby symbols. Every Rakefile is understood to have a default task called :default, which will be run if you don’t specify a task name

Source: Fulton, H & Arko, A 2015, The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Rake II

Eg:

```
$ rake          # execute the default task
$ rake mytask   # execute 'mytask'
```

Inside a Rakefile, we declare and create a task using the `task` method, passing it a symbol and a block:

```
task :mytask do
  # ...
end
```

The code inside the block, is referred to as an action

> Source: Fulton, H & Arko, A 2015, The Ruby Way: Solutions and Techniques in Ruby Programming, Third Edition Addison-Wesley Professional,

Rake III

Now, let's take a more concrete example. Imagine we have a C program named `myprog.c` with two other C files associated with it (each with its own header file). In other words, we have these five source files:

```
myprog.c
sub1.c
sub1.h
sub2.c
sub2.h
```

Rake IV

Rake provides a shortcut for declaring that a file can only be created if another file already exists. Let's begin by using that shortcut, the file method, to specify file dependencies:

Tools - Debugging

- **list 3-5**: Lists the specified lines of the program currently being worked upon
- **step:** steps through the program line by line
- **cont**: Runs the program without stepping. Execution will continue until the program ends, reaches a breakpoint, or a watch condition becomes true.
- **break 5**: Sets a breakpoint at a line number 5.
- **watch:** A conditional break point e.g.: when a variable gets a certain value
- **quit**: Exits the debugger.

Tools – Debugging watch and cont

Try: `ruby -r debug list_recursive_example.rb`

Source: Cooper, P 2016, Beginning Ruby From Novice to Professional. Apress (p. 214-215)

Tools – Debuggingstep

Type `ruby -r debug list_recursive_example.rb`

Tools – Debuggingbreak

Tools – Debuggingvariable

`v l` (variables local)

```
(rdb:1) v l
  list => ["a", "b", "c", "d", "e", "f"]
(rdb:1)
```

`v g` (variables local)

–Try with code to see output.

Type `ruby -r debug list_recursive_example.rb`

Ruby Profiler

Try it, type: `ruby -r profile insertion_sort.rb`

You should see:

Ruby Benchmarking

Ruby 3.0 Static Typing

Ruby 3.0 comes with a language called **RBS** to describe Ruby programs and a tool to work out the types involved in a Ruby program, called **TypeProf**. RBS allows programs to be checked using **steep**.

From: https://www.ruby-lang.org/en/news/2020/12/20/ruby-3-0-0-rc1-released/

RBS

RBS is a language to describe the types of Ruby programs.

Type checkers including TypeProf and other tools supporting RBS will understand Ruby programs much better with RBS definitions.

You can write down the definition of classes and modules: methods defined in the class, instance variables and their types, and inheritance/mix-in relations.

The goal of RBS is to support commonly seen patterns in Ruby programs and it allows writing advanced types including union types, method overloading, and generics. It also supports duck typing with *interface types*.

TypeProf

TypeProf is a type analysis tool bundled in the Ruby package.

Currently, TypeProf serves as a kind of type inference.

It reads plain (non-type-annotated) Ruby code, analyzes what methods are defined and how they are used, and generates a prototype of type signature in RBS format.

Here is a simple demo of TypeProf.

An example input:

```
# test.rb
class User
  def initialize(name:, age:)
    @name, @age = name, age
  end
  attr_reader :name, :age
end
User.new(name: "John", age: 20)
```

An example output:

```
$ typeprof test.rb
# Classes
```

```
class User
  attr_reader name : String
  attr_reader age : Integer
  def initialize : (name: String, age: Integer) -> [String, Integer]
end
```

You can run TypeProf by saving the input as “test.rb” and invoke a command called “typeprof test.rb”.

You can also try TypeProf online. (It runs TypeProf on the server side, so sorry if it is out!)

Try the following Ruby code in TypeProf:

Run RUBY

```
1
2
3 YEAR_TRUMP_ELECTED = 2016
4
5 def read_string(prompt)
6   puts prompt
7     # the analyser doesn't seem to know what type gets() returns
8   value = gets().to_s().chomp()
9 end
10
11 def read_boolean(prompt)
12     puts(prompt)
13   value = gets().chomp
14   case value
```

STEEP

STEEP will type check your code, perhaps using annotations to help:

https://github.com/soutaro/steep

See Also:

https://evilmartians.com/chronicles/climbing-steep-hills-or-adopting-ruby-types

Credit Task for this Week (12.1)

Task output should look like this:

Credit Task for this Week (12.1)

If there is a FAILED test, it will look like this:

Summary

We have looked at some tools you might use with Ruby.

We have seen how we might have used these tools to do some tasks during the course.

Eg: testing, performance measuring, debugging

Most languages have similar tools.