Ruby Tutorial

http://tryruby.org

Let's install some Ruby!

Windows:

http://rubyinstaller.org/

OS X:
brew install ruby

Linux:

```
sudo apt-get install ruby1.9.1
```

Hello World

Create a text file called `hello-world.rb` containing the following code:

```
puts 'Hello world'
```

Now run it

```
ruby hello-world.rb
```

You can also run the short "hello world" program without creating a text file. This is called a one-liner:

```
Ruby -e "puts 'Hello world'"
```

You can run this code with irb, but the output will look slightly different. `puts` will print out "`Hello world`", but irb will also print out the return value of puts – which is `nil`.

```
irb
>> puts "Hello world"
Hello world
=> nil
```

Ruby's interactive mode.

Syntax

Let’s say you have a class Person:

```
class Person
end

person = Person.new
person.name # => no method error
```

Obviously we never defined method name. Let’s do that:

```
class Person
  def name
    @name # simply returning an instance variable @name
  end
end

person = Person.new
person.name # => nil
person.name = “Dennis” # => no method error
```

We can read the name, but that doesn’t mean we can assign the name. Those are two different methods. Former called reader and latter called writer. We didn’t create the writer yet so let’s do that.

```
class Person
  def name
    @name
  end

  def name=(str)
    @name = str
  end
end

person = Person.new
person.name = ‘Dennis’
person.name # => “Dennis”
```

Awesome. Now we can write and read instance variable `@name` using reader and writer methods. But why waste time writing these methods every time?

```
class Person
  attr_reader :name
  attr_writer :name
end
```

Even this can get repetitive. When you want both reader and writer, just use accessor:

```
class Person
  attr_accessor :name
end

person = Person.new
person.name = "Dennis"
person.name # => "Dennis"
```

```ruby
class Person
  attr_accessor :name

  def greeting
    “Hello #{@name}”
  end
end

person = Person.new
person.name = “Dennis”
person.greeting # => “Hello Dennis”
```

Starting a new Ruby project:

```
cd ..
mkdir Ruby_uva
bundle gem Ruby_uva
cd Ruby_uva
git commit -m “Empty project”
rspec --init
```

Behavior Driven Development

Behavior-driven development combines the general techniques and principles of TDD with ideas from domain-driven design and object-oriented analysis and design to provide software developers and business analysts with shared tools and a shared process to collaborate on software development.

Although BDD is principally an idea about how software development should be managed by both business interests and technical insight, the practice of BDD does assume the use of specialized software tools to support the development process.

Where to start in the process

What to test and what not to test

How much to test in one go

What to call the tests

How to understand why a test fails

Acceptance tests should be written using the standard agile framework of a User story:

`As a [role] I want [feature] so that [benefit]`

Acceptance criteria should be written in terms of scenarios and implemented as classes:

`Given [initial context], when [event occurs], then [ensure some outcomes]`

Cucumber

we’re going to use a ‘Ruby off Rails’ course plain text way (gherkin) to write outlines

reads like a sales pitch

used together with rspec it’s called
‘outside in development’

every big feature > cucumber
unit tests > rspec / minitest / test:unit

Let’s create that `Gemfile`:

```
source “http://rubygems.org”

gem “rack”
gem “cucumber”
gem “rspec”
```

Run `bundle` **or** `bundle install`

```
mkdir features
mkdir features/step_definitions
mkdir features/support
mate features/student_can_submit_assignment.feature
```

The syntax of a Cucumber feature

```
Feature: Student Can Submit Assignments

As a student
I can submit my assignment
So I can prove my knowledge

Scenario: Student can submit an assignment
Given I am a student  # pre-condition
When I submit an assigment to my teacher # action
Then my teacher should have my assignment  # test
```

these by themselves don’t do anything

Run

```
bundle exec cucumber
```

in `features/step_definitions` **add** `students_assignment_steps.rb` **and paste in what your terminal so nicely provided you with:**

```
Given /^I am a student$/ do
pending # express the regexp above with the code you wish
you had
end

When /^I submit an assignment to my teacher$/ do
pending # express the regexp above with the code you wish
you had
end

Then /^my teacher should have my assignment$/ do
pending # express the regexp above with the code you wish
you had
```

Run `bundle exec cucumber` again and you'll see that because your `Given` is pending, it won't even bother to run your `When` and `Then`. Change your `Given` in:

```
Given /^I am a student$/ do
  @student = Student.new
end
```

Run `bundle exec cucumber` and it will give you `uninitialized constant Student`. We need another directory:

```
mkdir lib
vim lib/student.rb
```

Paste in:

```
class Student
end
```

In order for Cucumber to know about this new directory, open `features/support/` **and add a document called** `load_em.rb`**. Paste in:**

```
Dir[File.dirname(__FILE__) + "/../../lib/*.rb"].each{|f|
require f}
```

Run `bundle exec cucumber` **and you'll see** `Given I am a student` **is now green. Now that we've created the framework in Cucumber, we can go ahead and write our code TDD-style, writing tests in rspec. In** `students_assignment_steps.rb` **fill in your** `When`**:**

```
When /^I submit an assignment to my teacher$/ do
  @assignment = Assignment.new
  @teacher.submit_assignment(@student, @assignment)
end
```

Create `lib/assignment.rb` **and paste in:**

```
class Assignment
end
```

Open `student.rb` and add:

```
def submit_assignment(teacher, assigment)
end
```

under `class Student`. When running `bundle exec cucumber`, your test will return green. In `students_assignment_steps.rb` fill in your `Then`:

```
Then /^my teacher should have my assignment$/ do
  @teacher.assignment_for_student(@student).should eq(@as-
signment)
end
```

To solve the `undefined method 'assignment_for_student'` we need to add a teacher. Add:

```
Given /^I am a student$/ do
  @student = Student.new
  @teacher = Teacher.new
end
```

to your `students_assignment_steps.rb` and create a `teacher.rb` in your `/lib` folder:

```
class Teacher
 def assignment_for_student(student)
  end
end
```

We added `def assignment_for_student(student)` so you could avoid running Cucumber again and figuring out we don't have assignments yet either.

Now we're all ready to touch some rspec

```
mkdir spec
vim spec/teacher_spec.rb
```

In `teacher_spec.rb` **write:**

```
require_relative “../lib/teacher”
require “rspec”

describe Teacher do
  it “should store assignments” do
    student = stub
    assignment = stub
    subject.submit_assignment(student, assignment)
    subject.assignment_for_student(student).should
eq(assignment)
  end
end
```

Open `teacher.rb` and add

```
def initialize
  @assignments = {}
end
def submit_assignment(student, assignment)
  @assignments[student] = assignment
end
```

under `class Teacher`, and change

```
def assignment_for_student(student)
end
```

in

```
def assignment_for_student(student)
  @assignments[student]
end
```

When we run `bundle exec cucumber` we see some greens, some reds. Now would be a good time to initiate a git repository. Run:

```
git init
git status # to see a list of our untracked files
git add .
git commit -m "Add: Student can submit assignment"
```

Let's add another feature! Create `features/teacher_can_grade_assignment.feature` and write:

```
Feature: Teacher can grade assignment

As a Teacher
I can grade my students' assignments
So that they can know their knowlede level

@wip
Scenario: Teacher can grade assignment
  Given I have a student
  And They submitted an assignment
  When I grade the assignment
  Then the assignment has a grade
```

Now when we run `bundle exec cucumber --tags @wip` we avoid running both scenarios, as that might get noisy.

Create `features/step_definitions/teacher_grade_assignment.rb`. Pasting in the step definitions our terminal provides us with again, we can also add some classes already:

```
Given /^I have a student$/ do
 @teacher = Teacher.new
 @student = Student.new
 @assignment = Assignment.new
end

Given /^They have submitted an assignment$/ do
  @teacher.submit_assigment(@student, @assignment)
end

When /^I grade the assignment$/ do
  @teacher.record_grade(@assignment, 95)
end

Then /^the assignment has a grade$/ do
end
```

Open `lib/teacher.rb` and add

```
def record_grade(student, grade)
end
```

Open `spec/teacher_spec.rb` and add

```
describe “should record a grade” do
  it “can find an assignment” do
    student_a, assignment_a = stub(:student_a),
stub(:assignment_a)
    student_b, assignment_b = stub(:student_b),
stub(:assignment_a)
    subject.submit_assignment(student_a, assignment_a)
    subject.submit_assignment(student_b, assignment_b)
    subject.find_assignment(assignment_a).should
eq(assignment_a)
  end
end
```

Running the test you'll get an `undefined method` for `find_assignment`. Open `lib/teacher.rb` and add

```
def find_assignment(assignment)
  key = @assignment.select{|k,v| v == assignment}.first.first
  @assignment[key]
end
```

We then run `bundle exec cucumber --tags @wip` and start working on our `Then /^the assignment has a grade$/`. Open `teacher.rb` and change

```
def record_grade(student, grade)
end
```

in

```
def record_grade(student, grade)
  assignment = @assignments[student]
  assignment.grade = grade
  assignments[student] = assignment
end
```

And get rid off:

```
def find_assignment(assignment)
  key = @assignment.select{|k,v| v == assignment}.first.first
  @assignment[key]
end
```

Open `teacher_spec.rb` and replace

```
it "can find an assignment" do
  student_a, assignment_a = stub(:student_a),
stub(:assignment_a)
  student_b, assignment_b = stub(:student_b),
stub(:assignment_a)
  subject.submit_assignment(student_a, assignment_a)
  subject.submit_assignment(student_b, assignment_b)
  subject.find_assignment(assignment_a).should
eq(assignment_a)
end
```

with

```
it "should record the grade" do
  student = stub
  assignment = mock
  assignment.should_receive(:grade=).with(95)
  subject.submit_assignment(student, assignment)
  subject.record_grade(student, 95)
end
```

When we run `bundle exec cucumber --tags @wip` it won't actually show us the grade, we get an `undefined method` for `grade=`. Let's fix that.
Open `features/step_definitions/teacher_grade_assignment.rb` and change

```
When /^I grade the assignment$/ do
  @teacher.record_grade(@assignment, 95)
end
```

in

```
When /^I grade the assignment$/ do
  @teacher.record_grade(@student, 95)
end
```

Create `assignment_spec.rb` in your specs folder and write:

```
require_relative "../lib/assignment"

describe Assignment do
  it "should store a grade" do
    subject.grade = 60
    subject.grade.should eq(60)
  end
end
```

And finally add to `assignment.rb`

```
class Assignment
  attr_accessor :grade
end
```

Open `features/step_definitions/teacher_grade_assignment.rb` and change

```
Then /^the assignment has a grade$/ do
end
```

in

```
Then /^the assignment has a grade$/ do
  @teacher.assignment_for_student(@student).grade.should
eq(95)
end
```

Now when you run `bundle exec cucumber` and `rspec spec`, you should be all green!