Ruby Tutorial

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?

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

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

```ruby
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