COM1001 Introduction to Software Engineering

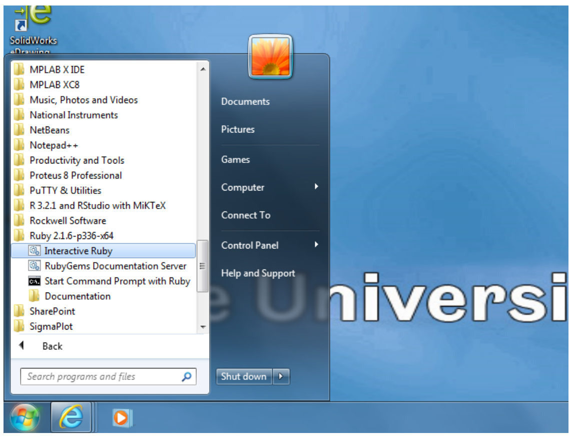
Ruby Introduction

October 7, 2015

Ruby concepts introduced in this lab: *Ruby Shell* · *Numbers* · *Strings* · *Expressions* · *Variables* · *Calling functions* · *Printing and reading* · *Scripts*

1 Ruby Interactive Shell

Ruby is installed on the CICS Windows desktop in the computers in the Diamond. (Later in the year, the computers will also support dual-booting to Linux.) For our first steps with Ruby, we will use the interactive shell provided by Ruby. To start the shell, click the 'Interactive Ruby' item under the 'Ruby 2.1.6' section on the Start menu:



The interactive Ruby shell allows you to type expressions, which are then evaluated and the result is printed. Try the following exercises to become familiar with Ruby expressions:

1. Ruby can evaluate numerical expressions. Type the following expression and press enter:

   ```
   5+5
   ```

   Ruby uses an arrow (=>) for responses to expressions. Try out expressions with other numbers and mathematical operators such as `*`, `-`, `/`.

2. A string is an ordered set of characters, on which Ruby can perform operations. To specify a string, enclose the text in double quotes. Now evaluate string expressions, such as `"Hello" + "World"`. Are other operators such as "-" also supported?

3. We can use the "*" operator to multiply strings with numbers. Try `"hello " * 5`

4. To perform actions on objects such as a string or a number, we can invoke *methods* on them. For example, we can count the number of letters in a string using the method `length` like in the following example: `"Sheffield".length`

5. We can reverse strings by invoking the `reverse` method. For example: `"Sheffield".reverse`

6. Try reversing a number (e.g. 123 to 321) using the `reverse` method – it does not work. However, we can convert the number to a string, and then reverse that, and convert it back to a number. Converting from strings to numbers is done using the `to_i` method, and from numbers to strings with the `to_s` method. Try `123.to_s.reverse.to_i`.

7. We can assign names to values if we want to reuse them; these are called *variables*. Note that we do not need to declare the type! Create a variable `x` and assign it the value 5: `x = 5`

8. Query the value of `x` by entering `x`

9. Now assign the value “`Hello`” to `x` and query the value of `x`. Would that be possible in Java?

10. Create a new variable `y` that contains the reverse of the string variable `x`. Can you find out what is the difference between using “`reverse`” and “`reverse!`” on a string variable?

11. `puts` is a function that takes a string as argument, and outputs it. Try out the following command: `puts "The reverse of #{x} is #{y}"`. Ruby will insert the values of variables `x` and `y` into this string. What happens if you change the value of `x` and `y` and then use the same `puts` command again? (You can go through the history of commands entered at the interactive Ruby shell by using the cursor up/down keys).

12. Now output the same sentence using `print`. What is the difference between `puts` and `print`?

13. The function `gets` can be used to read inputs from the user. It takes no arguments and returns a string that can be assigned to a variable. Use `gets` to read value, store it in a variable, and then use `print` to write it back. As you can see, the linebreak from the user input is included in the string returned from `gets`. To remove it, we can use the string method `chomp`, e.g. `gets.chomp`.

2 Ruby Scripts

Playing with the Ruby shell can be a nice way to explore aspects of the language, or to do quick calculations. However, if we write more complex program code, then we usually want to keep this for reuse later. Ruby can be used as a *scripting language*, which means that we can put lists of statements into a plain text file, and then ask the Ruby interpreter to execute all statements in the file.

For this exercise, use your favourite text editor (e.g. NotePad, Notepad++, jEdit) to edit Ruby scripts. You will also need a shell to run the Ruby script. You get this by selecting 'Start Command Prompt with Ruby' from the ‘Ruby 2.1.6’ entry in the start menu (see screenshot above). Using your editor, create a new text file, which will contain a list of Ruby statements – this is called a *script*. The script will contain a list of statements, one per line. Unlike in Java, statements do not need to be ended with a semicolon. Implement the following statements:

1. Output a prompt asking for the name (use `print`).

2. Read the input and store it in a variable `name` (use `gets` and `chomp`).

3. Now also ask the user for the year of birth, and store it in the variable `year`.

4. The variable `year` will now be a string, but in order to do calculations we need to convert it to a number. Use the `to_i` method of strings to make `year` an integer number.

5. Create a variable age, and calculate the number of years between the current year (just use the constant value 2015 for now) and the birth year given by the user.

6. Now produce a response (use `puts`) that states what name was entered and what the age is (see below for an example).

Save this script as `lab1.rb` and remember the path. On your command line, execute the script by invoking the Ruby interpreter, and passing it the full path name of your script:

```
ruby lab1.rb
```

If you do this correctly, you should be prompted for your name and year of birth, and receive your age as a response, for example like this:

```
What is your name? Gordon
What is your birth year? 1997
Your name is Gordon, and you are 18 years old.
```

As we used the constant value 2015 in our script, it will not work anymore next year. Wouldn't it be much better if our script would use the actual current year at the time it is run? Ruby has a standard library of helpful datastructures and algorithms, including a time-related library. To find out the documentation of the `Time` class, you can find the Ruby docs online at:

`http://ruby-doc.org/core-2.2.3/`

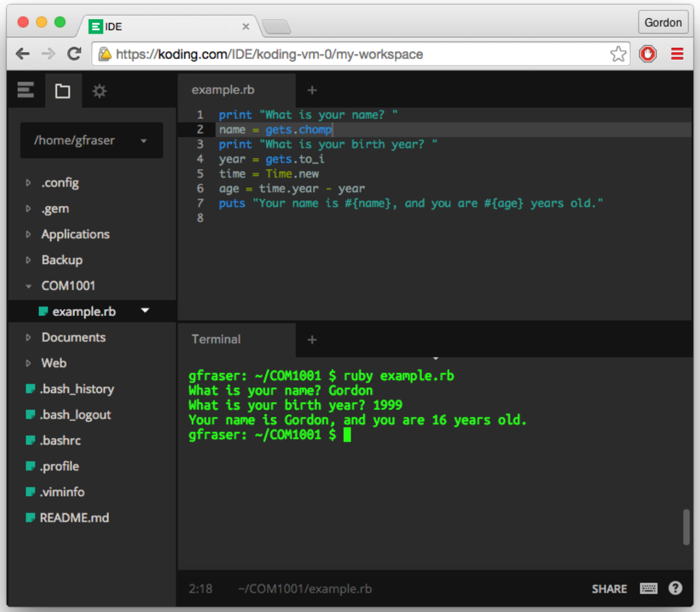
If you read through this documentation (`http://ruby-doc.org/core-2.2.3/Time.html`), you will find that you can create a new `Time` object set to the current time by using either `Time.new` or `Time.now` (both do the same). All objects come with *methods* you can invoke on them (see COM1003) by using ⟨variable_name⟩.⟨method_name⟩. For example, the `Time` class has a method `year` which returns the year of that object.

- Open your script in the editor
- Add a new variable `time` and assign it the value `Time.new`. This creates a new `Time` object.
- Retrieve the current year from `time` by invoking the `year` method.
- Calculate the age based on this year, rather than the hard coded value 2015.
- Save the script, and try running it.

(If this is all too easy for you, ask for the birth day and month as well, and calculate the age in days and months. Don't forget about leap years!)

3 Koding.org – Programming with Ruby Online (Optional)

You will be able to access Ruby via the CICS Windows Desktop on any computer at the University. To install Ruby on your own computer at home, you will find all required links to Ruby etc. on the module's MOLE page. However, if you're feeling adventurous, you may want to consider moving to an online IDE, such that you have exactly the same environment (and a much nicer one) on any computer you log in to. In particular, I suggest you take a look at `koding.org`, which offers you a free virtual machine running Linux, with Ruby, an online editor, and everything else you need for COM1001 installed:



```ruby
print "What is your name? "
name = gets.chomp
print "What is your birth year? "
year = gets.to_i
time = Time.new
age = time.year - year
puts "Your name is #{name}, and you are #{age} years old."
```

```
gfraser: ~/COM1001 $ ruby example.rb
What is your name? Gordon
What is your birth year? 1999
Your name is Gordon, and you are 16 years old.
gfraser: ~/COM1001 $
```

If you sign up to `koding.org` make sure to use the following link, so that we both get 500MB extra space :-)

```
https://koding.com/R/gfraser
```

4 First Steps in Ruby for Self Study (Optional)

There are plenty of online courses and material on Ruby available. If you want to do some more exercises of absolutely basic Ruby things, you could try these two online tutorials - they only require a web browser:

- http://tryruby.org/
- https://www.codecademy.com/tracks/ruby ("Introduction to Ruby" section)

You can find further links on the module's MOLE page.