COM1001 Introduction to Software Engineering

RubyMine Introduction

October 14, 2015

Ruby concepts introduced in this lab: *RubyMine IDE* · *Code comments* · *Modules*

1 RubyMine

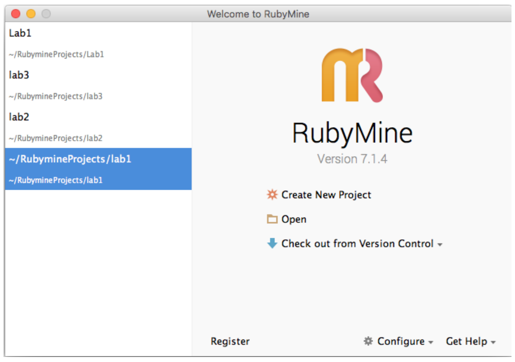
Working with a standard editor like Notepad (like we did last week) is ok for quick scripts, but if we want to do some serious programming, we will need something more suitable. Most programming is done in Integrated Development Environments (IDEs), which integrate all the tools one needs during development, such as a code editor, a compiler, a debugger, a graphical user interface (GUI) builder, code navigation tools, etc. There are several different IDEs for Ruby; in the computers in the lab we have installed RubyMine, which is one of the most popular IDEs for Ruby.

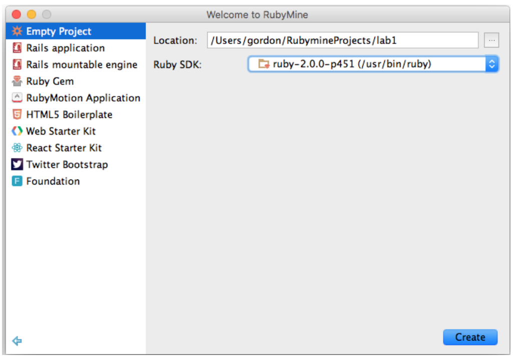
If you want to install RubyMine at home on your own computer, you can get a free student license at `https://www.jetbrains.com/student/` (you don't need this now in the lab).

1.1 Getting Started

1. Launch RubyMine. On Windows machines this is done by selecting "JetBrains RubyMine 7.1.4" from the "JetBrains" item in the start menu.

2. Create a new Ruby project by clicking on "Create New Project" (e.g. call it "lab2").

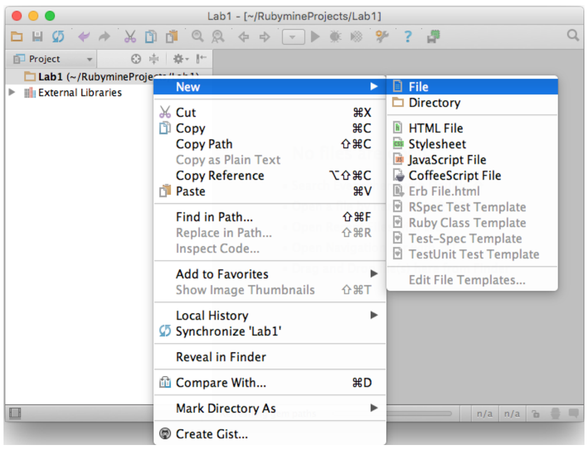




Location: /Users/gordon/RubymineProjects/lab1

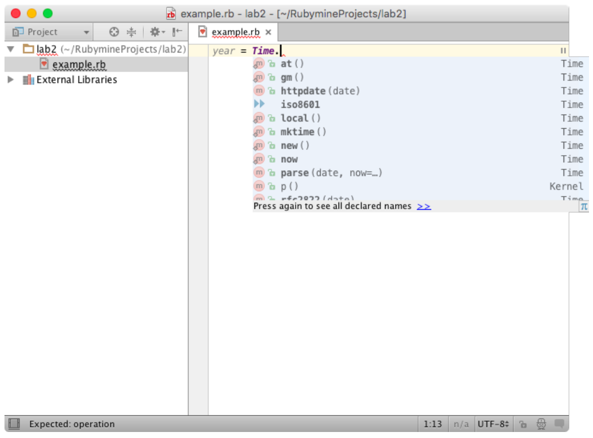
Ruby SDK: ruby-2.0.0-p451 (/usr/bin/ruby)

3. Create a new Ruby file (e.g., call it `example.rb`) in that project by right clicking on *lab2* in the project view, and selecting *New* → *File*.



1.2 Editing with Auto-completion

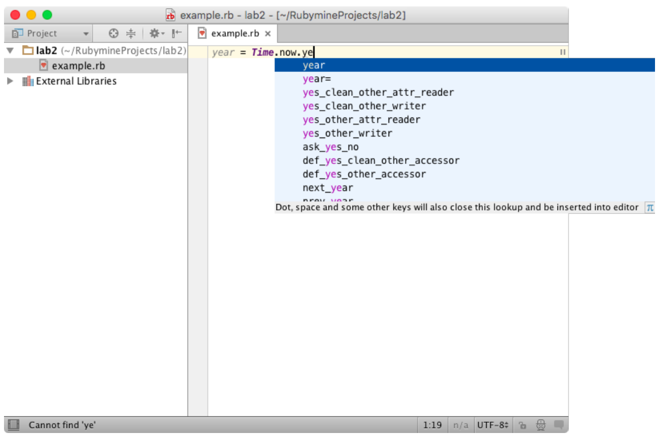
You can now edit the file `example.rb` in the editor. The editor provides a number of features that make coding much easier than writing the code in a basic editor like Notepad. One very useful feature is auto-completion. For example, let's create a new variable `year` and assign it the `year` value of `Time.now`. Type "`year = Time.`" (including the final dot), and RubyMine will open a menu with suggestions what should follow after the dot:



```
year = Time.
```

Expected: operation

Select `.now` (which creates a Time object initialised to the current time). Let's assume we want to access the `year` property of that object, so we add another dot, but RubyMine does not include `year` in that list. This is because RubyMine only shows us a small part of the possible completions. Hit Ctrl+Space to get a full list of completion. This list is rather long, but we can narrow it down by typing any part of the word (even characters somewhere in the middle), e.g.:


example.rb - lab2 - [~/RubymineProjects/lab2]
Project
example.rb
lab2 (~/RubymineProjects/lab2)
example.rb
External Libraries
year
year=
yes_clean_other_attr_reader
yes_clean_other_writer
yes_other_attr_reader
yes_other_writer
ask_yes_no
def_yes_clean_other_accessor
def_yes_other_accessor
next_year
Dot, space and some other keys will also close this lookup and be inserted into editor
1:19 n/a UTF-8

```
year = Time.now.ye
```

Cannot find 'ye'

You can use the up/down cursor keys to select any item of that list, end pressing enter will insert it into your file. If you just hit the tab key, it will insert the first item of the list.

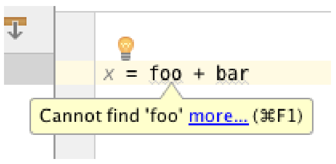
Auto-complete will also suggest you variable names. For example, create a variable `birthyear` that contains the birth year queried from the user, like in last week's exercise:

```
print "What is your birthyear? "
birthyear = gets.chomp.to_i
```

Now let's create a new variable `age` and assign it the value of `year` minus `birthyear`. We can do this without actually typing the full variable names; instead, just type `age = y<tab> - b<tab>` (where `<tab>` means you should hit the tab key on your keyboard).

1.3 Inspection and Warnings

RubyMine will point out problems in your code it can detect without running the code. For example, if you are trying to use variables that have not been defined, then the unknown variable names will be underlined with a wiggly line, and if you hover your mouse cursor over the variable name then you will get to see more information about the problem:

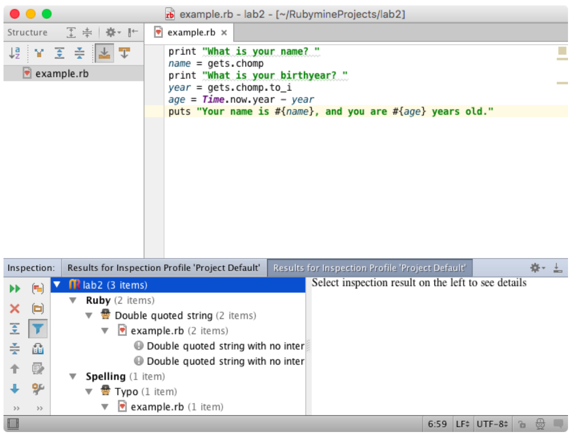


```
x = foo + bar
```

Cannot find 'foo' more... (⌘F1)

You will see similar warnings for other problems, for example if you do not close all parentheses, or there is a syntax error in the code you wrote.

RubyMine also offers a "Code inspector" that provides further information. For example, when applying the code inspector (Menu "Code", item "Inspect code...") to last week's program, it informs me that the use of double quotes (") in the `print` statements is redundant. Indeed, double quotes are only needed if you embed Ruby code that needs to be interpreted, using the #{name} syntax. In all other cases single quotes (') are sufficient. Inspect also shows spelling mistakes in strings — for example, "birthyear" isn't actually a word (whoops).



example.rb - lab2 - [~/RubymineProjects/lab2]

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

```
Inspection:  Results for Inspection Profile 'Project Default'  Results for Inspection Profile 'Project Default'
▼ lab2 (3 items)
    ▼ Ruby (2 items)
        ▼ Double quoted string (2 items)
            ▼ example.rb (2 items)
                Double quoted string with no inter
                Double quoted string with no inter
    ▼ Spelling (1 item)
        ▼ Typo (1 item)
            ▼ example.rb (1 item)
```

Select inspection result on the left to see details

6:59 LF UTF-8

1.4 Code comments

It is good practice to include (natural language) descriptions of your code, such that other people reading your code can more easily understand it. In Ruby, code comments start with the hashtag character (#), and anything that follows from this character to the end of the line will be ignored by the interpreter:

```
# This is a comment
```

If you want to add a comment that spans several lines, then each line needs to start with a #. Alternatively, you can use block comments:

```
=begin
This is a comment
that goes over several
lines.
=end
```

Comments can also be useful to "comment out" lines of code that you don't want to include when executing the code (for example if the code is broken, but you want to test the rest of the code, or if the code is obsolete but you want to keep it for later). In RubyMine, you can automatically toggle lines to be commented out by hitting the Ctrl+/ key, or by selecting "Comment with Line Comment" from the "Code" menu. If nothing is selected, then the current line will be changed to a comment, otherwise the entire selection will be changed to a comment. If you use the same command on a line that already is commented out, then RubyMine will automatically change it to regular code again. That is, you can toggle between commenting code in and out.

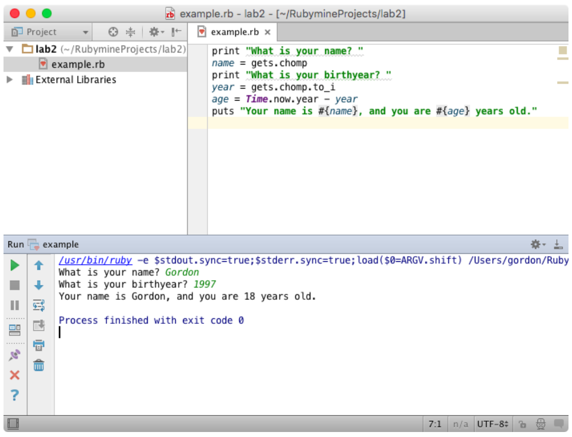
You can also find further related features in the "Code" menu item; for example, you can auto-format existing code, which fixes indentation etc. Once we start writing more complex code with loops, conditions, etc., you will also learn to appreciate that RubyMine will automatically indent code.

1.5 Running Ruby code

Complete the example code you have written to work like last week's example—here it is, as a reminder:

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

To execute this Ruby script, select the menu item "Run 'example' " from the "Run" menu, or from the context menu you get from right-clicking on `example.rb` in the project view. Once you do that, the bottom half of RubyMine shows you the program's output, and you can enter text like we did last week on the command line.

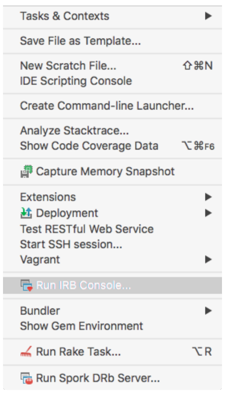

example.rb - lab2 - [~/RubymineProjects/lab2]
Project
lab2 (~/RubymineProjects/lab2)
example.rb
External Libraries
example.rb
Run
example
7:1
n/a
UTF-8

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

```
/usr/bin/ruby -e $stdout.sync=true;$stderr.sync=true;load($0=ARGV.shift) /Users/gordon/Ruby
What is your name? Gordon
What is your birthyear? 1997
Your name is Gordon, and you are 18 years old.

Process finished with exit code 0
```

If you want to use Ruby in interactive mode, you can also launch an interactive ruby shell using the command "Run IRB Console..." from the "Tools" menu:



1.6 Keyboard Shortcuts

RubyMine has plenty of further useful functionality, for example to navigate through a project with several files efficiently, to search and find relevant code, or to refactor existing code. Efficient users often tend to memorise many of the keyboard shortcuts so they do not have to navigate through the menus. You can find "cheat sheets" summarising the most important keyboard shortcuts for Windows and Linux here:

`https://www.jetbrains.com/ruby/docs/RubyMine_ReferenceCard.pdf`

If you are a Mac user, there is a different version of the cheat sheet:

`https://www.jetbrains.com/ruby/docs/RubyMine_ReferenceCard_Mac.pdf`

2 Coding Exercise

1. Create a new Ruby file called `Calculator.rb` in your current project (or create a new project, if you prefer).
2. In `Calculator.rb`, implement a simple command-line calculator as a Ruby script:
   a) First, the calculator should output a prompt (e.g., ">").
   b) Then, it should read an expression from the command line (`gets`) and store it in a variable.
   c) Make sure newlines are removed from that variable (use the function `chomp`).
   d) The `eval` function in Ruby takes a string as argument and evaluates it using the Ruby interpreter. We can use this to perform calculations by passing in expressions as strings. Evaluate the expression the user typed and store the result in a new variable. Recall that you can find details about the Ruby API in the RubyDocs:
   `http://ruby-doc.org/core-2.2.3/Kernel.html#method-i-eval`
   e) Print out the result in a nice way: "`The result of <original expression> is <result>`"
3. Execute the calculator by selecting *Run 'Calculator'*.
4. The calculator will now perform very simple calculations such as `5 + 5`, but to make it more interesting we would like to include more complex mathematical functions, such as `sin` or `cos`. In Ruby, these are contained in the `Math` module. Modules are a way of grouping together different methods, classes, and constants. All the details of the `Math` module are contained in the RubyDocs (`http://ruby-doc.org/core-2.2.0/Math.html`). We will learn how to create our own modules in a later lab.
5. Modules of the standard Ruby library can simply be used by calling functions in `Math` using the full namespace, e.g., `Math.cos`. This means that our calculator can already handle complex functions like `Math.cos(5) * 4`. (If we want to use modules that are not core libraries, we need to explicitly import them using the `require` command.)
6. It would be nicer if we didn't have to use the `Math` prefix every time we want to access a mathematical function. To achieve that, let's use the `include` command to import all functions defined in `Math` to our own namespace. Add `include Math` at the beginning of the calculator script. Now we can also evaluate expressions like `cos(5) * 4`.
7. Try the refactoring functionality in RubyMine to rename one of your variables, e.g., the one storing the expression entered by the user. Right click on the variable, select the "Refactor" menu, and from there select "Rename...". Now you can type a new name, and this will change not only the specific text you clicked on, but all occurrences of the variable.

(If this is too easy for you, try evaluating simple arithmetic expressions *without* using `eval`).