University College Dublin
An Coláiste Ollscoile, Baile Átha Cliath

SEMESTER I EXAMINATION – 2013/2014

COMP 41100

Exploring Programming in Ruby

Prof. A. Mille
Mr. J. Dunnion
Prof. M. Keane*

Time allowed: 2 hours

Instructions for candidates

Answer any FIVE Questions.
All Questions carry equal marks. Use of calculators is prohibited.

Instructions for invigilators

Use of calculators is prohibited.

1. Consider a function that takes the following array:

   ```
   [0, 1]
   ```

   and generates the next 10 Fibonacci numbers in the sequence to output the following:

   ```
   [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
   ```

   Now define two methods that each generate this sequence: (i) one called `fib1` that uses iteration and (ii) one called `fib2` that uses recursion.

2. Describe what Ruby does during *method lookup*, when an object calls a method (be it an instance or class method), how it searches for the method's definition and the conditions which lead to a `method_missing` error.

3. Define a class called `Turkey` (with three attributes, including one called `living` which can have the value `true/false`) and a subclass of it called `EuropeanTurkey` (with five attributes in total, one of which is called `size`).

   Create two methods for the `Turkey` superclass that are inherited by the class `EuropeanTurkey`, for which you should define one further method, called `check_price`; when invoked on an instance of EuropeanTurkey, `check_price` will return "too expensive" if the instance's size is "small" and the price is > 10 and "good value" if the instance's size is "big" and the price is < 10 (when instances have any other values for these attributes, "don't know" should be returned).

   Define a module called `KillThing` that has a method called `chop_it`, that will change the value of the `living` attribute to `false` when it is invoked on appropriate objects.

   Create a mixin, using the `KillThing` module, such that `Turkey` and `EuropeanTurkey` object-instances will be appropriately modified when the `chop_it` method is invoked on them.

   What is a mixin and why does Ruby use them?

4. Write an iterative method (using **`each`**, **`collect`** or **`select`**) – called `past_tense` – that will take an array of symbols (of any arbitrary length), such as:

   ```
   [:change, :kiss, :kick, :please]
   ```

   and produce the appropriate past-tense form for these regular verbs. So, for the above array, the method should return the array:

   ```
   [:changed, :kissed, :kicked, :pleased]
   ```

   Now, define a method – called `past_tense_sub` – that does the same thing using **`sub`** or **`gsub`**.

   Now define a method – called `count_letters` – that will return the array as an array showing the number of letters in each symbol-element of the array; for example, when dealing with the above original array it should return:

   ```
   [6, 4, 4, 6]
   ```

   Is it good practice to use symbols in this way? Briefly list some of the uses symbols are put to in Ruby.

5. Write a short explanatory paragraph on any ***four*** of the following, using appropriate examples: polymorphism, data abstraction, duck typing, modularity, inheritance in OOP.

6. Ruby on Rails makes use of the Model-View-Controller architecture pattern to organize the development of web-based applications. What are models, views and controllers?

   Write a short explanatory paragraph on each.

   Give three reasons why it might be a good idea to divide up web-based applications in this way.

7. What do the following evaluate to in Ruby:

i. `puts "dd"`

ii. `a = "foo"; puts a`

iii. `["a","b","c"].instance_of?(String)`

iv. `["a","b","c].instance_of?(Array)`

v. `class NewThing end; p NewThing.new`

vi. `[3, 4, 5, [6]].inject {|a, b| a < 4}`

vii. `["a","b","c"].each {|item|  puts item + "c"}`

viii. `["a1","b2","c3"].collect {|item| item[1].to_i.to_f}`

ix. `[[2,3],[3],[4,5]].length`

x. `[1,2,3,4,4,2,3,6,2,1,145,4,3,2].uniq`

xi. `Float.new`

xii. `"fooble ".concat("doodle")`

xiii. `["fooble"].concat(["doodle"])`

xiv. `["fooble"] << ["doodle"]`

xv. "fooblinggg".chomp.chop.chop

xvi. baDDarT.upcase

xvii. "apples_oranges_lemons".split(/ /)

xviii. "1234" <=> "12345"

xix. [6,3,2,1].inject{|x,y| x / y}

xx. Hash.new