COMP41100

EXPLORING PROGRAMMING IN RUBY

PRACTICAL TWO

29th September 2013

Andrew Doyle
Student Number: 12252388

University College Dublin

INTRODUCTION

This introduction serves to outline the format of this submission and provide a brief overview of the steps taken to answer each question. Each answer is described in more detail in the relevant section later in this report. Questions were posed on Stackoverflow by the Author (using the pseudonym Tom) which helped to create more useful solutions. This report also serves to demonstrate that the Author fully understands what is happening in each programme.

Format of this submission

This submission is contained within a zip folder. The **.rb** files are contained within relevant folders, such as **Question1**, **Question2** etc. This **README** report serves to explain in detail how the answers were coded and why the author chose to answer the question in a particular way. Suggested improvements are provided for Question 2 by the Author which could be implemented in future works.

Along with explaining the solution, the author includes code snippets and screenshots of the output to help illustrate the programme implementation. References are provided where relevant and a List of Figures is included in the Appendix.

Question 1

The aim in Question 1 is to implement a method in one file (**doggy.rb**) that uses methods found in two other files (**cheers.rb** and **i_like_dogs.rb**). A method **get_details** is defined in **doggy.rb** which calls on the methods found in other files(**print_cheers** and **print_like**), and which also calls on **print_new_name** which is defined in the same file.

The author asks the user for their name and their dog`s name. The user is thanked for their input and their dog`s name is renamed (simply by adding *"the doggy!!"* to the string) and output back to the user.

Question 2

In Question 2 a class is declared to store family member details and is instantiated five times. Methods called **parent?** **a**nd **child?** are defined to test whether a particular family member (instance of the class) are a parent (or child). To improve the author`s learning experience, and to reduce the amount of code in **family_members.rb**, the **parent?** and **child?** methods are defined in a separate class contained in **Family.rb**.

To improve the flexibility of the programme, the user is asked which family member they wish to check, and whether they wish to check if they are a parent or a child.

Question 3

In Question 3 an array is defined of the family members, with each individual`s details printed to the console by searching through the array using **each**. Furthermore the details are printed out within a string to provide a more descriptive result to the user.

Question 4

In Question 4 an array is defined of the family members, with each individual`s details printed to the console by searching through the array using **do**. Futhermore, the results are printed out in a tabular format, to improve on the visual appearance of Question 3.

QUESTION ONE

In **doggy.rb**, **require_relative** is used to include the other files (**cheers.rb** and **i_like_dogs.rb**) where methods are defined. Note how the **.rb** suffix is not required within the single quotes:

```
#doggy.rb

#include other .rb files where methods are defined
require_relative 'cheers'
require_relative 'i_like_dogs'

#This method calls on two methods defined in other files
def get_details
  puts "What is your name?:"
  usersname = gets.chomp
    puts "What is your dogs name?:"
    dogsname = gets.chomp
    print_cheers(usersname)  #from cheers.rb
    print_like(dogsname)     #from i_like_dogs.rb
    print_new_name(dogsname) #defined in this file
end

#This method is called in the get_details method
def print_new_name(doggy) #doggy is dynamically input by the user
    dogs_full_name = "\n" + doggy + " the doggy!!! (Yes..that was lame)"
    puts "Your doggies new name is: "
    puts dogs_full_name
end

get_details
```

In the **get_details** method the user is asked what their name is and the response is stored in the **username** variable using **gets.chomp**. “**chomp**” is required to remove carriage return characters such as *n* and *r*. Similarly the user is asked for their dog`s name which is stored in the variable **dogsname**.

The **print_cheers** method is then called (passing in the **usersname** variable as a parameter). This method is defined in **cheers.rb**; it outputs a thank you message to the user (with the user`s name included within the message):

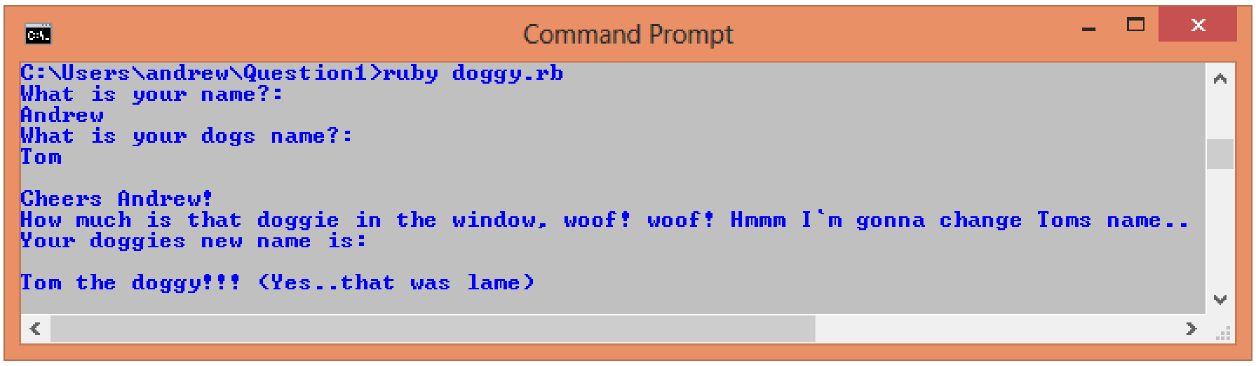
```
#cheers.rb

#called in get_details method in doggy.rb
def print_cheers(usersname)
    puts "\nCheers " + usersname +"!"
end
```

Next, the **print_like** method is called (passing in the **dogsname** variable as a parameter). This method is defined in **i_like_dogs.rb**; it outputs a message to the user informing them that the dog`s name will be changed (with the dog`s name included within the message):

```
#i_like_dogs.rb

#called in get_details method in doggy.rb
def print_like(dogsname)
    puts "How much is that doggie in the window, woof! woof! Hmmm I`m gonna change " +
dogsname + "s name.."
end
```

Defining methods in separate files is beneficial as it makes the code clearer and less cluttered. Figure 1.1 shows a sample run of the programme at the Windows Command Prompt.

Figure 1.1 – doggy.rb



```
C:\Users\andrew\Question1>ruby doggy.rb
What is your name?:
Andrew
What is your dogs name?:
Tom

Cheers Andrew!
How much is that doggie in the window, woof! woof! Hmmm I`m gonna change Toms name..
Your doggies new name is:

Tom the doggy!!! (Yes..that was lame)
```

QUESTION TWO

In **doggy.rb**, **require_relative** is used to include the **family.rb** file, where the **parent?** and **child?** methods are defined. The class **FamilyMember** is declared, which extends the **Family** class (from **family.rb** where **parent?** and **child?** are defined).

Using the method **attr_accessor** allows you to create both a reader and a writer for a given attribute at the same time. The **initialize** method is declared (which Ruby searches for whenever a new object is created). This allows the variables to be initialized during the instantiation of the class.

```ruby
#family_members.rb

require_relative 'family'

class FamilyMember < Family #allows the use of methods in the Family class from
family.rb
  #create reader and writer in one go:
  attr_accessor :name, :sex, :type, :role, :age
  def initialize (name, sex, type, role, age) #parameters for initialization during
instantiation of class
    @name = name
    @sex = sex
    @type = type
    @role = role
    @age = age
  end

end
```

A hash called **fm** is then declared. As outlined by Burd (2007, p.102) "*A hash is like an array, except that a hash`s indices aren`t necessarily numbers. More precisely, a hash is a collection of key/value pairs. Each pair is called an entry*".

Thomas, Fowler and Hunt(2013, p.47) expand on this when they say: "*while you index arrays with integers, you index a hash with objects of any type*". Furthermore, they outline how "*When you store a value in a hash, you actually supply two objects – the index, which is normally called the key, and the entry to be stored with that key*".

The instances of the **FamilyMember** class are created using the **new** keyword, whilst the variable values are passed in as parameters in conjunction with the **initialize** method previously discussed.

```ruby
# Below, a hash is created called fm; instances of the class are then instantiated
within the hash elements
fm = {}
fm[1] = FamilyMember.new('Andrew','Male', 'Child', 'Son' , '27' )
fm[2] = FamilyMember.new('Bill','Male', 'Parent', 'Father' , '63' )
fm[3] = FamilyMember.new('Samantha','Female', 'Parent', 'Mother' , '62' )
fm[4] = FamilyMember.new('Thomas','Male', 'Child', 'Dog' , '10' )
fm[5] = FamilyMember.new('Samantha', 'Female', 'Child', 'Dog' , '4' )
```

The **check_details** method is declared which reads input from the user and acts accordingly. In this case **define_method** is required, rather than just **def**. If **def** was used, an *undefined local variable or method* error would occur for **fm**. This is because **def** is a method definition that creates an isolated scope where variables defined inside are not accessible outside and vica versa.

Conversely, **define_method** creates an instance method, whereby variables from outside the method are accessible. Inside the **check_details** method a **repeat** variable is declared (this is used to check whether or not the user has chosen to exit the programme). The **while** loop is executed when **repeat** is not zero (the user can set it to zero at the end of a programme run if they wish to exit).

The user is asked what family member they wish to check, with the answer stored in **family**. This is converted to an integer using **to_s** and stored in **num**. A message is then output to the console informing the user that the chosen family member is being checked. The **#{}** construct allows the output of a variable within a string.

The user is then asked whether they want to check if the chosen family member is a child (choose 0) or a parent (choose 1), with the choice stored in **choice** after being converted to an integer. A **case statement** is then created to carry out different tasks based on the users choice.

When the user chooses 0 (child) **then** the **child?** method is called on the key of **fm**(the key is provided by the user – previously stored in **num**). However this only takes place **if** the given key is present in the hash (this is checked using **fm.key?(num)** which will return true or false). If it returns false (i.e. the user chose a value other than 0 or 1) the **else** statement is implemented whereby the user is warned they must choose 0 or 1. The second **when** block calls the **parent?** method in the same way as the first **when** block did.

The user is asked whether they want to check another family member, with their choice converted to an integer and stored in **repeat**; which as previously discussed is the test condition for the **while loop**. After the method is declared it can then be called, by simply writing **check_details**.

```
# Below, the check_details method asks the user which family member they wish to check,
and the parent? or child? methods are called
define_method :check_details  do #allows variables to be accessed outside of this scope
(as opposed to def)

  repeat = ''   #variable to check whether or not to repeat loop

  while repeat != 0
    puts "\nWhat family member do you want to check? choose 1 to 5"
    family = gets.chomp     #chomp removes carriage return characters
    num = family.to_i       #convert to integer

    puts "Checking family member #{num}"

    puts "\nDo you want to check if family member #{num} are a child or an parent?"
    puts "Child  | 0"
    puts "Parent | 1"

    userchoice = gets.chomp
    choice = userchoice.to_i

    case
    when choice == 0
      if fm.key?(num) then  #check if given key is present in the hash
        fm[num].child?      #call the child? method on the given instance
      end
    when choice == 1
      if fm.key?(num) then
        fm[num].parent?
      end
    else
      puts 'Be Careful - You must choose 0 or 1. 0 for child or 1 for parent'
    end

    puts "\nDo you want to check another family member?"
    puts "Exit          | 0"
    puts "Check Another | 1"
    repeat = gets.chomp
    repeat = repeat.to_i    #users choice on whether to repeat program - this is the
condition for the loop

  end

end

check_details               #method called
```

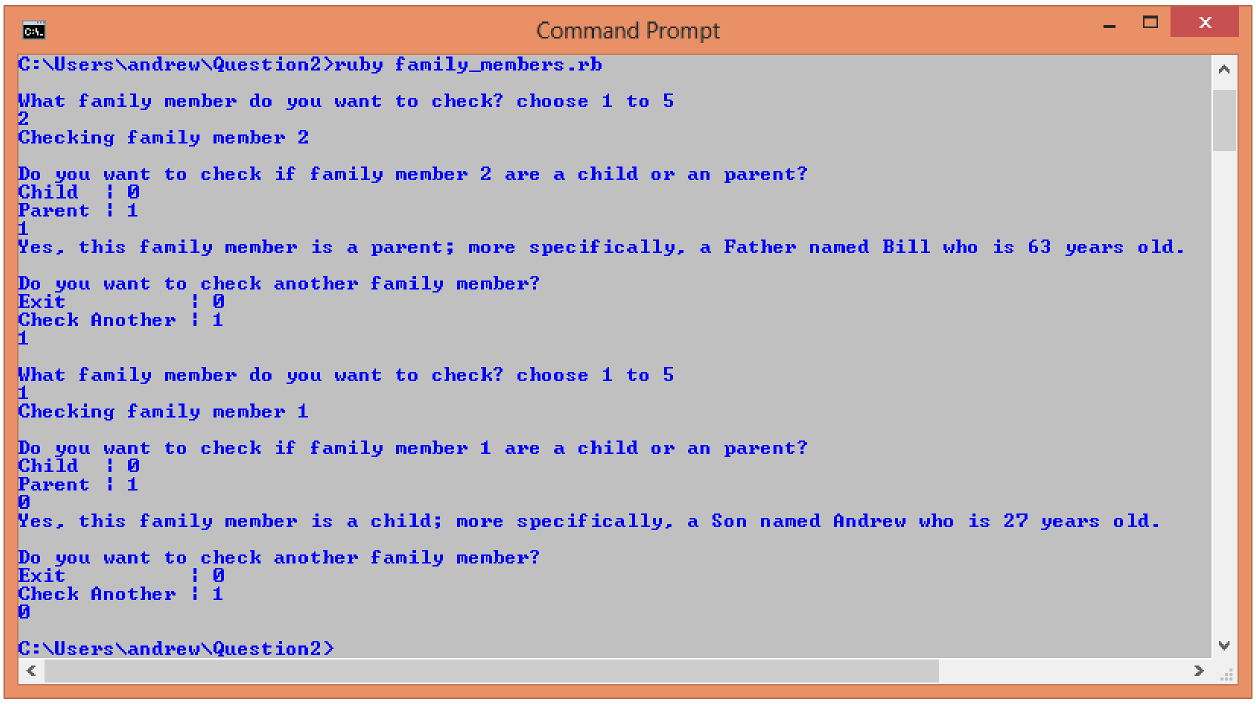
As previously outlined, the **parent?** and **child?** methods are declared in the **Family** class in **family.rb** (shown below). Instance variables are denoted using the **@** symbol, for example, **@type**. Traditionally, a method ending with a question mark (**?**) in ruby should return **true** or **false**.

In this application, in order to provide a more meaningful response, the author has expanded on this. A message is output to the user which is different depending on the result. If the **parent_or_child if-statement** returns **true**(i.e. it equals 'Parent'), then the user is informed that the family member is indeed a parent; with other details such as role, name and age also output in the message.

```ruby
#family.rb

class Family    #methods in this class are called in the FamilyMember class declared in 
family_members.rb
  def parent?
    parent_or_child = @type #denotes fm instance variables
    role = @role
    age = @age
    name = @name

    if parent_or_child == 'Parent'
    then puts "Yes, this family member is a parent; more specifically, a #{role} named 
#{name} who is #{age} years old."  #message if true
    else puts "No, this family member is not a parent; it is a child..more 
specifically, a #{role} named #{name} who is #{age} years old."
    end
  end

  def child?
    parent_or_child = @type
    role = @role
    age = @age
    name = @name

    if parent_or_child == 'Child'
    then puts "Yes, this family member is a child; more specifically, a #{role} named 
#{name} who is #{age} years old."
    else puts "No, this family member is not a child; it is a parent, more 
specifically, a #{role} named #{name} who is #{age} years old."
    end
  end

end
```

Figure 2.1 on page 8 shows the output from a sample run of the programme.

Figure 2.1 – Sample run of family_members.rb



Command Prompt

```
C:\Users\andrew\Question2>ruby family_members.rb

What family member do you want to check? choose 1 to 5
2
Checking family member 2

Do you want to check if family member 2 are a child or an parent?
Child  | 0
Parent | 1
1
Yes, this family member is a parent; more specifically, a Father named Bill who is 63 years old.

Do you want to check another family member?
Exit          | 0
Check Another | 1
1

What family member do you want to check? choose 1 to 5
1
Checking family member 1

Do you want to check if family member 1 are a child or an parent?
Child  | 0
Parent | 1
0
Yes, this family member is a child; more specifically, a Son named Andrew who is 27 years old.

Do you want to check another family member?
Exit          | 0
Check Another | 1
0

C:\Users\andrew\Question2>
```

<u>Suggested Improvements:</u>

A useful feature would be to display an error message and restart loop if user specifies a family member other than 1 – 5. Currently no error message is displayed; the user is simply asked would they like to check another family member. Traditionally a method ending with a question mark should return a Boolean value. The programme could be altered to adhere to this norm.

QUESTION THREE

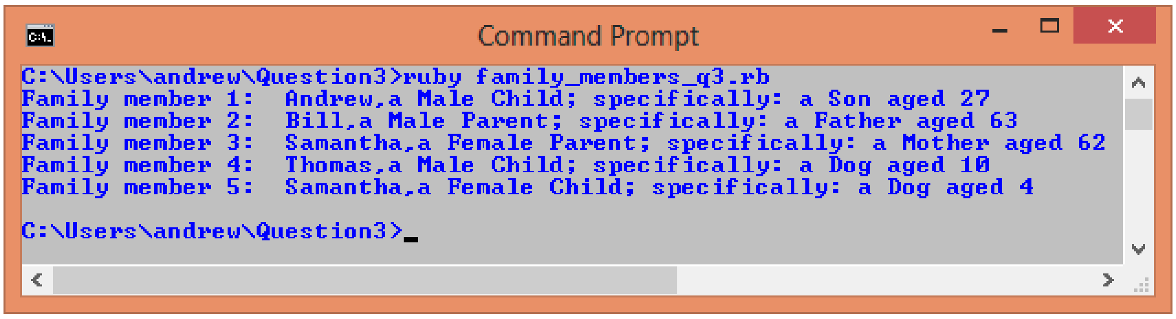
In Question Three, the class is declared in the same manner as Question Two. On this occasion an array is created, rather than a hash. The instances of the class are created and pushed into the array named **fm** using the **<<** operator. The difference between **<<** and **push** is that **push** can accept many arguments whilst **<<** accepts only one argument.

```
#family_members_q3.rb

class FamilyMemberQ3
  #create reader and writer in one go:
  attr_accessor :name, :sex, :type, :role, :age
  def initialize (name, sex, type, role, age) #parameters for initialization during
instantiation of class
    @name = name
    @sex = sex
    @type = type
    @role = role
    @age = age
  end

end

# Below, an array is created called fm; instances of the class are then instantiated
within the array elements
fm = []

# << pushes one instance at a time into the fm array
fm << FamilyMemberQ3.new('Andrew','Male', 'Child', 'Son' , '27' )
fm << FamilyMemberQ3.new('Bill','Male', 'Parent', 'Father' , '63' )
fm << FamilyMemberQ3.new('Samantha','Female', 'Parent', 'Mother' , '62' )
fm << FamilyMemberQ3.new('Thomas','Male', 'Child', 'Dog' , '10' )
fm << FamilyMemberQ3.new('Samantha', 'Female', 'Child', 'Dog' , '4' )

# 1 signifies label of index on first iteration, member refers to each instance
fm.each.with_index(1) { |member, index|
  puts "Family member #{index}:  #{member.name},a #{member.sex} #{member.type};
specifically: a #{member.role} aged #{member.age}"
}
```

The instance method **each_with_index** from the **Enumerator** class is used to iterate through the **fm** array and output its contents. The method is appended to **fm** (The **1** signifies the label to give the **index** on the first iteration; which is iterated by one for each instance) whilst **member** refers to each instance.

The instance variable names are appended to **member** in the method block where a message is output to the user with each instance`s details. Figure 3.1 shows a run of the programme.

Figure 3.1 – Sample run of family_members_q3.rb



```
C:\Users\andrew\Question3>ruby family_members_q3.rb
Family member 1:  Andrew,a Male Child; specifically: a Son aged 27
Family member 2:  Bill,a Male Parent; specifically: a Father aged 63
Family member 3:  Samantha,a Female Parent; specifically: a Mother aged 62
Family member 4:  Thomas,a Male Child; specifically: a Dog aged 10
Family member 5:  Samantha,a Female Child; specifically: a Dog aged 4

C:\Users\andrew\Question3>_
```

QUESTION FOUR

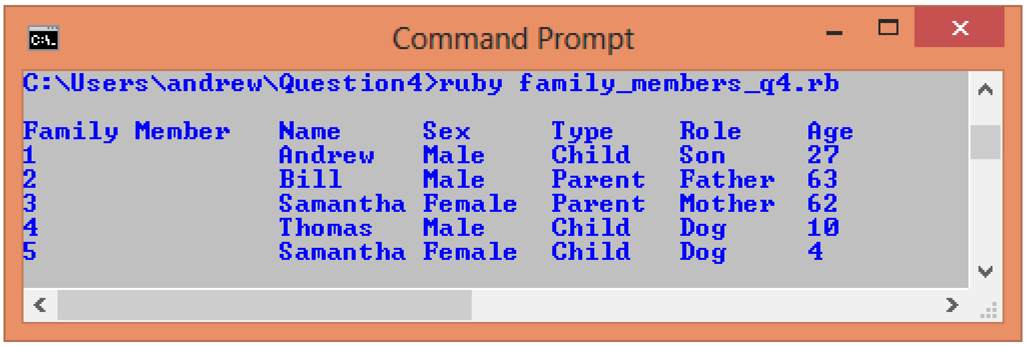
In Question Four, the class is declared in a less code-heavy manner. The parameters for the **initialize** method are represented by ***args**, rather than listing each one as was done in previous questions. The instance variables are linked to **args** in the **initialize** method body as shown in the code snippet below.

The array is instantiated and pushed into an array as before. On this occasion, the author decided to output the results in a tabular format; which is more presentable for the user. To specify the width of each column the variable **format** is declared where the width is denoted using %-15s (15 characters wide – the family member column). The column headings are output before the array search using **format %**.

The array is searched using **each_with_index** in a similar manner to Question 3, however on this occasion **do** is used to perform the search. **I** refers to the array index, and **member** refers to each instance in the array. In the method body, **I** is iterated by one to ensure that 1 is entered for *"family member"* instead of 0 (which is the array index).

```
#family_members_q4.rb

class FamilyMemberQ4

  attr_accessor :name, :sex, :type, :role, :age
  def initialize (*args) #args refers to parameters passed during instantiation as
shown below
    @name, @sex, @type, @role, @age = args
  end

end

# Below, an array is created called fm; instances of the class are then instantiated
within the array elements
fm = []

# << pushes one instance at a time into the fm array
fm << FamilyMemberQ4.new('Andrew','Male', 'Child', 'Son' , '27' )
fm << FamilyMemberQ4.new('Bill','Male', 'Parent', 'Father' , '63' )
fm << FamilyMemberQ4.new('Samantha','Female', 'Parent', 'Mother' , '62' )
fm << FamilyMemberQ4.new('Thomas','Male', 'Child', 'Dog' , '10' )
fm << FamilyMemberQ4.new('Samantha', 'Female', 'Child', 'Dog' , '4' )

# Formats the desired output for each column
format = '%-15s %-8s %-7s %-7s %-7s %s'
puts "\n"
puts format % ['Family Member', 'Name', 'Sex', 'Type', 'Role', 'Age']

# i signifies label of index on first iteration, member refers to each instance
fm.each_with_index do |member, i|
  puts format % [ i+1, member.name, member.sex, member.type, member.role, member.age ]
end
```

Figure 4.1 – Sample run of family_members_q4.rb



```
C:\Users\andrew\Question4>ruby family_members_q4.rb

Family Member   Name     Sex     Type    Role    Age
1               Andrew   Male    Child   Son     27
2               Bill     Male    Parent  Father  63
3               Samantha Female  Parent  Mother  62
4               Thomas   Male    Child   Dog     10
5               Samantha Female  Child   Dog     4
```

REFERENCES

1. Burd,B. (2007). Ruby on Rails For Dummies. Indianapolis, Indiana. United States of America: Wiley Publishing Inc.

2. Stackoverflow.com (2013). User 2402476 Questions. [Online Forum] Retrieved 29th September 2013, from http://stackoverflow.com/users/2402476/tom?tab=questions.

3. Thomas, D., Fowler,C. , Hunt, A. (2013). Programming Ruby 1.9 & 2.0 The Pragmatic Programmer`s Guide. United States of America: The Pragmatic Bookshelf.

LIST OF FIGURES