

**COMP41100**

**EXPLORING PROGRAMMING IN RUBY**

**PRACTICAL SIX**

**31<sup>st</sup> October 2013**

**Andrew Doyle**  
**Student Number: 12252388**

## Table of Contents

PART ONE .....	2
PART TWO .....	3
PART THREE .....	4
PART FOUR .....	5
LIST OF IMAGES .....	7

## PART ONE

### Task:

Get 5 regular emails and 5 advance---fee fraud emails (aka spam). Convert them all into text files and then turn each into an array of words (split may help here). Then use a bunch of regular expressions to search the array of words looking for keywords to classify which files are spam or not. If you want to get fancy you could give each array a spam---score out of 10.

### Solution:

Five sample spam emails were retrieved online (such as Nigerian spam, soldier in Iraq with \$25m etc) together with five regular emails (such as funny joke emails, regular emails etc.). The files were read in and split into an **array** of words:

```
file1 = File.open("spam1.txt", "rb")
file1_contents = file1.read
file1 = file1_contents.split(' ')
```

The words are then read into a **hash**:

```
freqs1 = Hash.new(0)
file1.each { |word| freqs1[word] +=1 }
```

A list of typical spam words is specified:

```
bad_words = %w{business fund funds account transfer money}
```

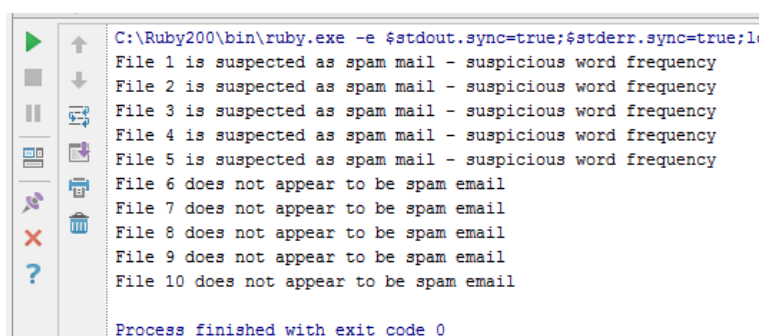
A method checks whether any of the spamable words appear more than twice (with Iraq also added as a check to see if it appears more than once). The **any?** method from the **Enumerable** module is used to check each word in the email; the **is\_spam1** method as shown below returns true if the count of words is as specified in the condition.

```
is_spam1 = freqs1.any? do |word, freq|
  (freq > 2 && bad_words.include?(word)) || (word == 'Iraq' && freq > 1)
end
```

Finally, a message is output to the user which informs them whether or not the email has been classified as spam:

```
if is_spam1
  puts "File 1 is suspected as spam mail - suspicious word frequency"
else
  puts "File 1 does not appear to be spam email"
end
```

Figure 1.1 – Spam Check



```
C:\Ruby200\bin\ruby.exe -e $stdout.sync=true;$stderr.sync=true;1
File 1 is suspected as spam mail - suspicious word frequency
File 2 is suspected as spam mail - suspicious word frequency
File 3 is suspected as spam mail - suspicious word frequency
File 4 is suspected as spam mail - suspicious word frequency
File 5 is suspected as spam mail - suspicious word frequency
File 6 does not appear to be spam email
File 7 does not appear to be spam email
File 8 does not appear to be spam email
File 9 does not appear to be spam email
File 10 does not appear to be spam email

Process finished with exit code 0
```

## PART TWO

### Task:

Do some scraping using Nokogiri on an arbitrarily selected web page from somewhere.

### Solution:

```
require 'nokogiri'
require 'open-uri'
require 'pp'
doc =
  Nokogiri::HTML(open("http://www.conserveturtles.org/seaturtleinformation.php?page=whyca
reaboutseaturtles"))
pp doc
```

Figure 2.1 – Partial output from scrape

```
C:\Ruby200\bin\ruby.exe -e $stdout.sync=true;$stderr.sync=true;load($0=ARGV.shift) "C:/Users/andrew/Dropbox/CS/Semester 3/Ruby/Week 7/p
#(Document:0x12998dc {
  name = "document",
  children = [
    #(DTD:0x12a57b0 { name = "HTML" }),
    #(Comment " Header start "),
    #(Element:0x12a9f20 {
      name = "html",
      children = [
        #(Element:0x12a8870 {
          name = "head",
          children = [
            #(Element:0x12b2dc4 {
              name = "meta",
              attributes = [
                #(Attr:0x12b247c { name = "name", value = "keywords" }),
                #(Attr:0x12b2470 {
                  name = "content",
                  value = "sea turtles, turtles, conservation, satellite tracking, research, Costa Rica, endangered species, Tortuguero,
                })]
              },
            #(Element:0x13fb264 {
              name = "meta",
              attributes = [
                #(Attr:0x13fb048 { name = "name", value = "description" }),
                #(Attr:0x13fb03c {
                  name = "content",
                  value = "Sea Turtle Conservancy is dedicated to protecting sea turtles through research, education, advocacy and prote
                })]
              },
            #(Element:0x1405680 {
              name = "meta",
              attributes = [
                #(Attr:0x14054ac { name = "property", value = "og:title" }),
                #(Attr:0x14054a0 {
                  name = "content",
                  value = "Sea Turtle Conservancy :: Learn About Sea Turtles"
                })]
              },
            #(Element:0x140472c {
              name = "meta",
              attributes = [
                #(Attr:0x1404570 {
                  name = "name",
```

## PART THREE

### Task:

Take a fairly complex HTML document and draw the tree of Nokogiri elements it expands into, so you know how it parse represents the document

### Solution:

The webpage [www.skysports.com/football](http://www.skysports.com/football) is parsed as XML, and instructed to ignore whitespace only nodes during parsing (using `&:noblanks`). `xhtml` is used to specify pretty printing parameters.

```
require 'nokogiri'
require 'open-uri'
doc = Nokogiri::HTML(open("http://www.skysports.com/football"))

document = Nokogiri::XML(doc, &:noblanks)
puts document
puts doc.to_xhtml(indent: 3, indent_text: "  ")
```

Figure 3.1 – Pretty printing partial output

```
C:\Ruby200\bin\ruby.exe -e $stdout.sync=true;$stderr.sync=true;load($0=ARGV.shift) "C:/Users/andrew/Dropbox/CS/Semester 3/Ru
<?xml version="1.0"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  <head class="no-js">
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Football Games, Results, Scores, Transfers, News | Sky Sports</title>
    <meta charset="UTF-8" />
    <meta name="description" content="Sky Sports Football - Live games, scores, latest football news, transfers, results,
    <meta name="keywords" content="Sky, Sports, football, news, live, scores, video, breaking, fixtures, results, team, li
    <meta name="Revisit-after" content="2 days" />
    <meta name="application-name" content="SkySports" />
    <meta name="msapplication-tooltip" content="SkySports" />
    <meta name="msapplication-window" content="width=1024;height=768" />
    <meta name="msapplication-task" content="name=Home Page;action-uri=http://www.skysports.com;icon-uri=http://www.skyspo
    <meta name="msapplication-task" content="name=Score Centre;action-uri=http://live.skysports.com;icon-uri=http://www.sk
    <meta name="msapplication-task" content="name=Events Centre;action-uri=http://www.skysports.com/eventscentre;icon-uri=
    <meta name="msapplication-task" content="name=Video Clips;action-uri=http://www.skysports.com/video/clips;icon-uri=htt
    <script src="/static/preload-min-39cb496eeb2d8.js" type="text/javascript"></script>
    <script type="text/javascript">
<![CDATA[
      var redirectUrl = 'http://m.skysports.com/football';
      bskyb.redirectToMobile(redirectUrl);
    ]]>
  </script>
  <script type="text/javascript">
<![CDATA[
      var skyIdDomain = "skyid.sky.com";
    ]]>
  </script>
<!-- CDN hosted jQuery -->
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js"></script>
<!-- Local jQuery fallback if CDN version is down -->
  <script>
<![CDATA[window.jQuery || document.write('<script src="/static/jquery-1.6.4-7dfe4932e9cd2.js">\x3C/script>')]>
  </script>
  <link rel="stylesheet" href="/static/site-min-6c3a0775d47e0.css" type="text/css" media="all" title="default" />
  <link rel="stylesheet" href="/static/refresh-2013-0332e43632028.css" type="text/css" media="all" title="default" />
  <script>
<![CDATA[
    (function() {
```

## PART FOUR

### Task:

Define your own class that illustrates the differences between public, private and protected methods. It should not be anything like the ones I did.

### Solution:

To demonstrate the difference between public, private, and protected methods a scenario is created for a bank; namely a piggy bank.

A class is declared, called **Q4MethodAccessModifiers** which can be initialized with the attribute **cash\_left** (cash left in the piggy bank):

```
class Q4MethodAccessModifiers
  attr_reader :piggy_bank_confirmed
  protected  :piggy_bank_confirmed
  attr_accessor :cash_left

  def initialize(cash_left)
    @cash_left = cash_left
  end
end
```

Methods are declared to transfer money in between **piggy\_bank1** and **piggy\_bank2** (two separate accounts in the main piggy bank):

```
def method_protected_debit(piggybank, amount)
  piggybank.cash_left -= amount
end

def method_protected_credit(piggybank, amount)
  piggybank.cash_left += amount
end

def method_protected_transfer(amount)
  method_protected_debit(@piggy_bank1, amount)
  method_protected_credit(@piggy_bank2, amount)
end
```

To test the different access modifiers, private and protected methods are created; with the access modifiers for each method specified in a single block of code:

```
public      :method_public_transfer, :method_protected_transfer, :method_private_transfer
private    :method_private_credit, :method_private_debit
protected  :method_protected_credit, :method_protected_debit
```

A subclass is also created to test the inheritance of the methods:

```
class MoneyMovesSubclass < MoneyMoves

  def initialize(piggy_bank1, piggy_bank2)
    @piggy_bank1 = piggy_bank1
    @piggy_bank2 = piggy_bank2
  end
end
```

Instances of the main piggy bank are declared:

```
cash_behind_the_piano = Q4MethodAccessModifiers.new(50000)
cash_in_my_wallet = Q4MethodAccessModifiers.new(50)
```

The values are tested before any method implementation:

```
puts "\nTesting the values before any methods implemented: \n\n"
puts "Cash behind the piano: #{cash_behind_the_piano.cash_left}"
puts "Cash in my wallet: #{cash_in_my_wallet.cash_left} "
```

An instance of the **MoneyMoves** class is created and the protected\_method\_transfer is tested, with the new values output afterwards:

```
puts "\nTesting the implementation of the methods - an instance of MoneyMoves is
created and a value of 20 passed into the protected method: \n\n"
move_money_test = MoneyMoves.new(cash_in_my_wallet, cash_behind_the_piano)
move_money_test.method_protected_transfer(20)
puts "Cash behind the piano: #{cash_behind_the_piano.cash_left}"
puts "Cash in my wallet: #{cash_in_my_wallet.cash_left} "
```

An instance of the **MoneyMovesSubclass** subclass is created and the private\_method\_transfer is tested (this works because whilst the debit and credit methods used in the transfer method are private, the transfer method itself is public):

```
move_money_test_subclass =
MoneyMovesSubclass.new(cash_in_my_wallet, cash_behind_the_piano)

puts "\nTesting the implementation of the methods - an instance of MoneyMovesSubclass
is created and a value of 20 passed into the private method: \n\n"
move_money_test_subclass.method_private_transfer(20)
puts "Cash behind the piano: #{cash_behind_the_piano.cash_left}"
puts "Cash in my wallet: #{cash_in_my_wallet.cash_left} "
```

**Figure 4.1 – Method access modifier test run**

```
C:\Ruby200\bin\ruby.exe -e $stdout.sync=true;$stderr.sync=true;load($0=ARGV.shift) "C:/Us

Testing the values before any methods implemented:

Cash behind the piano: 50000
Cash in my wallet: 50

Testing the implementation of the methods - an instance of MoneyMoves is created and a va

Cash behind the piano: 50020
Cash in my wallet: 30

Testing the implementation of the methods - an instance of MoneyMovesSubclass is created

Cash behind the piano: 50040
Cash in my wallet: 10

Process finished with exit code 0
```

## LIST OF IMAGES

Figure 1.1 – Spam Check.....	2
Figure 2.1 – Partial output from scrape.....	3
Figure 3.1 – Pretty printing partial output.....	4
Figure 4.1 – Method access modifier test run.....	6