CIT 483/583 Lab –OpenURI & Nokogiri

**Instructions**

Open and save this file in any MS word-compatible format as Lab07\_*Firstname*\_*Lastname*.<ext> and place your answers in that document. Do a **Save-As** and retain all of my content. Keep the document safe in case your submission fails, or you discover an error prior to the due date and wish to re-submit. Submit your document to the Lab07 dropbox in Canvas. The due date and any other pertinent information are noted in the Canvas item.

This lab should be completed on students.cs.nku.edu as a reference implementation and as a means of testing your answers.

**Place the answers in or immediately following each question and make sure your answers stand out from the questions by using a different font color.**

## OpenURI & Nokogiri

1. OpenUri can open any URI, which it treats as an IO object. It is interesting to see what happens if the resource is not a plain text file. Perform the following Linux command line interactions and capture any output, **but please abbreviate long blocks of text to the last few lines and don't worry if a command produces no output**.

$ url="https://www.nku.edu/content/dam/nkuhome/images/homefeatures/griffin-hall-fb.jpg"

$ ruby -ropen-uri -e "puts open('$url')" > picture.jpg

$ wget $url

$ file \*jpg

(a) Inspect the output of the Ruby one-liner directly (don't redirect to a file).

$ ruby -ropen-uri -e "puts open('$url')"

This explains why picture.jpg is not really a graphics file. It is a File object, and we need to process it as a file to see its contents.

(b) We need to tweak the command to generate the *contents* of the file.

$ ruby -ropen-uri -e "puts open('$url').**read**" | more

Hit return a few times and then Ctrl-C. Clearly, this is not line-oriented text.

(c) This seems to be binary data, so let’s recreate the first attempt to generate a .jpg file.

$ ruby -ropen-uri -e "puts open('$url').read" > picture.jpg

$ cmp -b -l picture.jpg griffin-hall-fb.jpg

(d) Since puts adds a newline, let's see if that is the one-byte difference.

$ ruby -ropen-uri -e "print open('$url').read" > picture.jpg

$ ls -l \*jpg

$ cmp -b -l picture.jpg banner.jpg # no output means they are the same

1. Perform the following interactions in irb. Capture the output of each command. **Abbreviate any output that exceeds 8-10 lines**.

$ irb --simple-prompt

>> require 'open-uri'

require 'nokogiri'

>> doc = open ("https://inside.nku.edu/informatics/departments/computerscience.html")

>> page = Nokogiri::HTML(doc)

>> page.css('img').each { |img| puts img['src'] }

>> page.css('div').collect { |div| div['class'] }.uniq

>> page.css('a').collect { |a| a['href'] }.each { |href| puts href }; p

You will need to adapt the sample commands given above and/or in the slides to obtain your answers, but they will be similar. Show the commands you use to obtain the answers.

(a) How many total anchor tags ("a") are in the document?

(b) How many unique href's are used within those anchor tags?

(c) How many image ('img') tags are in the document?

(d) How many unique image sources ('src') are used?

1. Nokogiri can also process XML files. If you use the same IRB session, you will have the required libraries, but if not, make sure you add those libraries to a new session. **Make sure that the file master\_scoreboard.xml from an earlier lab is in the directory in which IRB is running.** As usual, capture the output of all commands, but abbreviate long blocks to the last few lines.

$ irb --simple-prompt

>> require 'open-uri'

require 'nokogiri'

>> raw = open("master\_scoreboard.xml")

>> raw.readpartial(80) # read 80 chars

>> doc = Nokogiri::XML(open('master\_scoreboard.xml'))

>> doc.xpath('//game').each do |p|

puts p.attributes['id']

end

>> doc.xpath('//winning\_pitcher').each do |p|

puts p.attributes['last']

end

As mentioned in the slides, the slop method can generate objects/methods for elements, but in a limited way. Nested elements are treated as if they are children of higher-level elements.

>> puts doc.games # should error out

>> doc = Nokogiri::XML(open('master\_scoreboard.xml')).slop!; p

>> puts doc.games.class # games now returns an element

>> puts doc.games.game

>> doc.games.game.each do |p|

puts p.attributes['id']

end; p

1. Copy ~mccordt/find\_links.rb find\_links.rb on students.cs.nku.edu to a location that you can write to. For reference, here is the content:

#!/usr/bin/ruby

# file: find\_links.rb

require 'open-uri'

require 'nokogiri'

if ARGV.length == 0

$stderr.puts "Must specify a URL"

exit 1

end

ARGV.each do |arg|

open(arg) do |f|

# Display connection data

puts "#"\*25 + "\nConnection: '#{arg}'\n" + "#"\*25

[:base\_uri, :meta, :status, :charset, :content\_encoding,

:content\_type, :last\_modi\_ed].each do |method|

puts "#{method.to\_s}: #{f.send(method)}" if f.respond\_to? method

end

# Display the anchor links

Nokogiri::HTML(f).css('a').each do |anchor|

href = anchor['href']

puts href

end

end # open(arg)

end # ARGv.each

(a) Run the script with the following argument and show the first 8-10 lines and the last 8-10 lines of output:

$ ruby find\_links.rb https://inside.nku.edu/informatics/departments/computerscience.html

(b) At least one of the hrefs displayed by the script is a relative path (/content/dam/informatics/docs/griffinhall/dept-research-page.pdf).

For the href beginning with a '/', we should prepend the host's URL. Add the following line (just before the call to Nokogiri::HTML(f).css) to obtain the necessary host url text.

# Derive the host base url

host\_url = /^(.\*\.nku\.edu)\//.match(f.base\_uri.to\_s)[1]

puts "host\_url: #{host\_url}"

Now modify the code to prepend this text to hrefs that begin with a '/' to make them fully-qualified URLS. Use regular expressions to check the href and append the host\_url accordingly.

Nokogiri::HTML(f).css('a').each do |anchor|

href = anchor['href']

# type your added code here

puts href

end

(c) Add the following code to find those hrefs that may be problematic for users.

Nokogiri::HTML(f).css('a').each do |anchor|

href = anchor['href']

# code from (b) here

# puts href # comment out the pus to generate less output

if href =~ /^https?:\/\//

begin

puts href unless open(href).status[0] == "200"

rescue Exception => e

puts "ERROR: #{e} -- #{href}"

end

end

List any problematic links you found.

1. Perform the following irb interactions to locate the source code for OpenURI's open method. Capture the output of each command.

$ irb --simple-prompt

>> method(:open).owner

>> method(:open).source\_location

>> require 'open-uri'

>> method(:open).owner

>> method(:open).source\_location

## Optional: Sockets

**If you did not do this kind of activity in CIT 383 and want to see Ruby's socket interface in action, this section shows you how the Ruby Socket interface works.**

For this part, you should open two terminals, both connecting to students.cs.nku.edu.

Create the following Ruby program. (You may copy it from students.cs.nku.edu @ ~mccordt/simple-server.rb or cut and paste into vi.)

#!/usr/bin/ruby

# simple-server.rb

# demo for a Ruby echo server

require 'socket'

# set up optional args

ip\_re = /^(?:\d{1,3}\.){3}\d{1,3}$/

ip, port = ['0.0.0.0', 1024 + rand(20000)]

ip = ARGV[0] if ip\_re === ARGV[0]

port = ARGV[1].to\_i if /\d+/ === ARGV[1]

[:ip, :port].each do |var|

puts "Variable: #{var}=#{eval var.to\_s}"

end

server = TCPServer.new(ip, port)

# loop infinitely, processing one incoming message per iteration

socket = server.accept

loop do

request = socket.gets

STDERR.puts "Received: #{request}"

socket.print "Request was: #{request}"

break if ( !request || request.chomp == 'bye' )

end

# cleanup

socket.close

1. For each of the following record the output of the command. If it fails, record the exception and backtrace. (If it succeeds, use Ctrl-C to kill the process, as it is in an infinite loop listening for clients.) Since there are multiple students doing the lab, **add the month (1-12) plus the day of the month of your birthday to the number 20072** used in the samples, or use another random number in the range 0-100.

$ ruby simple-server.rb 127.0.0.1 22

$ ruby simple-server.rb 127.0.0.1 20072

$ ruby simple-server.rb students.cs.nku.edu 20072

$ ruby simple-server.rb localhost 20072

$ ruby simple-server.rb 327.5.5.5 20072

1. For the next section run the commands in Courier font and answer questions a thru e. Since there are multiple students doing the lab, **add the month (1-12) plus the day of the month of your birthday to the number 78004** used in the samples, or use another random number in the range 0-100 to add to 78004.

$ ruby simple-server.rb 127.0.0.1 78004

[If the command throws an exception similar to 'initialize': Address already in use increment the number by **the month (1-12) plus the day of the month of your birthday as described above**.]

1. What appears to be wrong with this set of connection arguments above?
2. Despite that, did it succeed?
3. In a second terminal while the first command is still running, enter the following command. Try to determine which port the server is actually listening on and enter it as the answer:

$ lsof -iTCP -sTCP:LISTEN | grep $USER

1. Enter the following command (changing the original number 78004 as needed if you found that the original port was blocked) and record the result:

$ ruby -e 'p 78004 % 2\*\*16'

1. Given all of that, what does Ruby do when a requested port number exceeds 216 -1?
2. For the next section, run the following paired commands on two terminals to observe whether the processes succeed in communicating. [**Wherever the port number is specified, you may need to modify the port numbers if that port is in use as described earlier in the lab.**]

If you don't specify a port, the script chooses one at random. You can use the port displayed on terminal "A" to determine the port to specify on terminal "B". You *may* need to use Ctrl-C to kill one or more processes on terminals "A" or "B". Put "Success" or "Failure" as the answer to each item a thru c and follow the instructions for letter d.

a)

(TERM A) $ ruby simple-server.rb 127.0.0.1 9294

(TERM B) $ cat /etc/passwd | netcat 127.0.0.1 9294

b)

(TERM A) $ ruby simple-server.rb 127.5.0.1 3241

TERM B) $ cat /etc/passwd | netcat 127.5.0.1 3241

c)

TERM A) $ ruby simple-server.rb localhost 5404

TERM B) $ echo "Hello" | netcat localhost 5404

d) Capture the actual output of TERM B for this question.

Type the individual lines into TERM B separated by the Enter key.

TERM A) $ ruby simple-server.rb

NOTE: Variable: port number output of TERM A to determine the port to connect to from TERM B

Variable: port=<whatever it shows>

TERM B) netcat 172.28.100.236 <port from TERM A>

asdf

hi

Hello

Goodbye

bye