CSE/ISE 337: Scripting Languages
Stony Brook University
Programming Assignment #2
FALL 2024

Assignment Due: 30th October, Wednesday, by 11:59 PM (EST)

After completion of this programming project, you should be able to:

- Design and implement scripts in Ruby
- Extend existing Ruby classes with your functionality
- Design and implement an object-oriented system

Instructions

- Take time to read the problem descriptions carefully.
- Pay attention to the function names and class names. Incorrect names may lead to heavy penalties; could also result in your submission not being graded
- You need to submit four files. Zip them. Your <<Lastname_Firstname+SUBID>> should be the name of your zipped file.
  - **First file array.rb**
  - **Second file rgrep.rb**
  - **Third file contains_virus.rb**
  - **Fourth file vehicle.rb**
- You can use the test cases provided in the document or create your test cases.
- Important→ Zip all files and submit the zipped distribution
- Use the sample cases test cases to understand the problem and to test your code. However, we will use additional test cases to test your code.
- Create your test cases to thoroughly test your code.
- **All code must be written in Ruby and properly commented. Five points will be deducted for poorly commented code.**
- Tentative rubric:
  - 5% for software engineering: comments and coding style
  - 30% for your test cases. You need to write 20 test cases including the one given in the documents as examples (5 test cases for each problem). TAs will judge the quality of your test cases as well.
  - 65% for test cases prepared by the TAs.

Problem 1 (30 points) (Anusha Avulapati)

The *Array* class in Ruby has numerous methods. Of these methods, two methods – *[]* and *map*, are of particular interest to us.

The *[]* method is used to obtain the value at a particular position in an array. For example, if *a* is an array such that a = [1,2,34,5], then *a[0] = 1, a[1] = 2,* and so on. If we call the *[]* method with an index that is out of bounds, then *[]* returns the *nil* value.

The *map* method, when invoked on an instance of class *Array*, applies the code block associated with *map* to every element in the array and then returns a new array. For example, if *a* is an array such that *a = [1,2,34,5]* and we call *map* on array *a* with the code block *{ |x| x.to_f }*, then *a.map { |x| x.to_f }* will result in a new array, a' = *[1.0,2.0,34.0,5.0]*.

We want to change the behavior of the *[]* and *map* methods in the *Array* class. For the *[]* method, we want to return a default value '\0' instead of *nil* when an out-of-bounds index is passed to the *[]* method. If an index that is not out of bounds is passed as an argument to *[]*, then the method should return the value at that index. For example, for an array *a = [1,2,34,5]*, *a[2]* and *a[-2]* should return 34 since both indices are in-bound. However, *a[5]* or *a[-6]* should return the default value '\0' since both of those indices are out of bounds.

For the *map* method, we want to change its behavior such that it can be called with an optional sequence argument. When the sequence argument is provided, *map* should apply the associated code block to only those elements that belong to the indices in the sequence. If an invalid sequence is provided, then an empty array should be returned. If the sequence of indices is not provided, then the *map* method should default to its original behavior, that is, apply the associated code block to all elements in the array. Consider the following example:

```
b = ["cat", "bat", "mat", "sat"]
b.map(2..4) { |x| x[0].upcase + x[1,x.length] } b.map { |x| x[0].upcase + x[1,x.length] }
```

In the above example, the first call to *map* will result in the array ["Mat", "Sat"] because the associated code block is only applied to elements in positions 2,3, and 4 of array b. On the other hand, the second call to *map* will result in the array ["Cat", "Bat", "Mat", "Sat"] since no sequence was provided. Hence the original *map* function, which applies the associated code block to all elements in the array was called.

If the sequence provided to the *map* method contains indices in the array that are out of bounds, then the new array should not contain any value for these indices. For example, if we provide the sequence (2..10) to our *map* method, as shown below, the new array should be ["Mat", "Sat"].

```
b = ["cat", "bat", "mat", "sat"]
b.map(2..10) { |x| x[0].upcase + x[1,x.length] }
```

Sample Test Cases

```
a = [1,2,34,5]
puts a[1] # 2
puts a[10] # '\0'
p a.map(2..4) { |i| i.to_f} # [34.0, 5.0]
p a.map { |i| i.to_f} # [1.0, 2.0, 34.0, 5.0]

b = ["cat", "bat", "mat", "sat"]
puts b[-1] # "sat"
puts b[5] # '\0'
p b.map(2..10) { |x| x[0].upcase + x[1,x.length] } # ["Mat", "Sat"]

p b.map(2..4) { |x| x[0].upcase + x[1,x.length] } # ["Mat", "Sat"]

p b.map(-3..-1) { |x| x[0].upcase + x[1,x.length] } # ["Bat", "Mat", “Sat”]

p b.map { |x| x[0].upcase + x[1,x.length] } # ["Cat", "Bat", "Mat", "Sat"]
```

Reference

https://ruby-doc.org/core-2.4.2/Array.html

Submission Guideline

The file name should be array.rb

The file should be organized as follows:

```
class Array
```

```
  def [](index)
  …
  end

  def map(sequence = nil, &block)
  …
  end
end

#Your own test cases start here
a = [1,2,34,5]
puts a[1] # 2
puts a[10] # '\0'
p a.map(2..4) { |i| i.to_f} # [34.0, 5.0]
p a.map { |i| i.to_f} # [1.0, 2.0, 34.0, 5.0]

b = ["cat", "bat", "mat", "sat"]
…
```

Problem 2 (40 points) (Suyi Chen and Manthan Singh)

grep is a command line utility tool for searching plain-text data sets for lines that match a regular expression. We want to develop a grep-like utility using Ruby. Our grep-like utility will be called rgrep. On a unix-based system we should be able to use rgrep by using the following command:

$ path/to/rgrep/rgrep.rb

On a non unix-based system (e.g., Windows) we will use the normal ruby command to call our utility

$ ruby path/to/rgrep/rgrep.rb

Just like the grep utility takes a filename and offers numerous options to search the filename, our rgrep utility will also take a filename and based on the provided option combinations will return the search result. The following are the options it supports:

• -w <pattern>: treats <pattern> as a word and looks for the word in the filename. It returns all lines in the filename that contains the word.

• -p <pattern>: treats <pattern> as a regular expression and searches the filename based on the regular expression. It returns all lines that match the regular expression.

• -v <pattern>: treats <pattern> as a regular expression and searches the filename based on the regular expression. It returns all lines that do not match the regular expression.

• -c <pattern>: can only be used in conjunction with options –w, -p, or –v. For each conjunction, it returns the number of lines that match the pattern. Note : Although -p is the default setting, -c should be called with -p. Otherwise, it is not acceptable.

• -m <pattern>: can only be used in conjunction with options –w, or -p. For each conjunction, it returns the matched part of each line that matches the pattern. Note : Although -p is the default setting, -m should be called with -p. Otherwise, it is not acceptable.

Usage

A user of rgrep should provide a file name (fully qualified path if not in the current directory) followed by an option or a valid combination of options, and the pattern that will be used to search in the provided file.

Any combination of options other than the ones listed above should result in an error message “Invalid combination of options”

If any option other than the ones listed above is provided, an error message “Invalid option” should be reported.

If the combination of options is valid, then the order of the options does not matter.

If a filename or pattern is missing display “Missing required arguments” The default option is -p.

Sample Test Cases

Consider a file “test.txt” that contains the following lines:

101 broad road

101 broad lane

102 high road

234 Johnson Street

Lyndhurst Pl 224

$ ./rgrep.rb

Missing required arguments

$ ./rgrep.rb test.txt

Missing required arguments

$ ./rgrep.rb test.txt -f

Invalid option

$ ./rgrep.rb test.txt –v –m ‘\d’

Invalid combination of options

```
$ ./rgrep.rb test.txt –w road

101 broad road 102 high road



$ ./rgrep.rb test.txt –w –m road

road

road

$ ./rgrep.rb test.txt –w –c road

2

$ ./rgrep.rb test.txt –p ‘\d\d’

101 broad road

101 broad lane

102 high road

234 Johnson Street

Lyndhurst Pl 224

$ ./rgrep.rb test.txt –p –c ‘\d\d’

5

$ ./rgrep.rb test.txt –v ‘^\d\d’

Lyndhurst Pl 224

$ ./rgrep.rb test.txt –v –c ‘^\d\d’

1

$ ./rgrep.rb test.txt ‘\d\d’
```

101 broad road

101 broad lane

102 high road

234 Johnson Street

Lyndhurst Pl 224

Submission Guidelines

The file name should be rgrep.rb

There are no specific requirements for formatting. However, please make sure your rgrep.rb can be run in the terminal using commands such as "./rgrep.rb test.txt –p '\d\d'" or "ruby ./rgrep.rb test.txt –p '\d\d'"

Regarding test case submission, for problem 2 only, please snapshot the results shown in the terminal after you test on your test cases, and zip them together into your <<Lastname_Firstname+SUBID>>.zip

You can test your code with the following shell script, and it is easy to replace the test cases with yours.

```
#!/bin/bash

# Test case 1: Missing required arguments
ruby rgrep.rb

# Test case 2: Word search
ruby rgrep.rb test.txt -w road

# Test case 3: Regex search with count
ruby rgrep.rb test.txt -p -c "\d\d"

# Test case 4: Inverted search
ruby rgrep.rb test.txt -v "^\d\d"
```

```
# Test case 5: Invalid option
ruby rgrep.rb test.txt -f
```

Additional Notes

If you want to change your ruby file to an executable add #!/usr/bin/env ruby to the first line of your file. Then, change the permissions of your file with the following command:

```
$ chmod +x rgrep.rb
```

However, this is not required. It is perfectly ok to run your program normally.

Problem 3 (20 points) (Yoosung Jang)

Imagine you are in a game where a virus is spreading rapidly, and your task is to quarantine the infected area by installing walls.

The world is modeled as an m x n binary grid infected.

The value of the grid will determine if a place is infected or not.

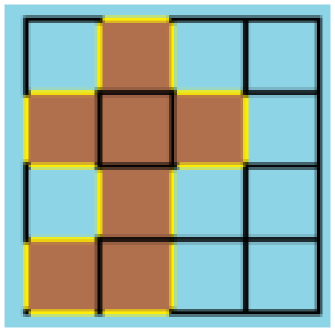
If the value isInfected[i][j] == 0 represents uninfected cells, and if isInfected[i][j] == 1 represents cells contaminated with the virus.

A wall **(and only one wall)** can be installed between any two 4-directionally adjacent cells, on the shared boundary.

Our goal is to stop the contamination of the virus, so we add walls to contain the Virus. The Virus does not spread to its neighboring regions. So, our Goal is to find the number of walls built to contain the virus. Please note: In this question, the main constraint is that there is only one area where the virus will exist. This means that all the regions where the virus is will share an adjacent cell.

Sample Test Case 1 ->

Input: isInfected = [[0,1,0,0],[1,1,1,0],[0,1,0,0],[1,1,0,0]]



Output: 16

Explanation -> The Brown color squares are where the virus is at. We make a wall around each virus region and return the result. (The Walls are colored Yellow).

Note -> You need to include the walls on the Border as well; as seen by the Yellow color walls in the image. In the cell with row 1 and column 1, even though it contains a virus, we don't have to make any walls for it. So we don't include this in the final answer.

You need to write a function :

```
def contain_virus(isInfected)
 ...
end
```

```
isInfected = [[0,1,0,0],[1,1,1,0],[0,1,0,0],[1,1,0,0]]

# Call the function and store the result in a variable

result = contain_virus(isInfected)

# Print the result

puts "Number of walls needed: #{result}"
```

Submission Guidelines

The file name should be contains_virus.rb
The file should be organized as follows:

```
def contain_virus(isInfected)
  …
end

# Your own test cases start here
isInfected = [[0,0,1,0],[1,1,0,1],[0,0,0,0],[1,0,1,0]]
puts isInfected
result = contain_virus(isInfected)
puts "Number of walls needed: #{result}" # Number of walls needed: 10
```

Problem 4 (20 points) (Sai Manne)

- Create a **superclass** called **Vehicle** for your **MyCar class and MyTruck class.** Create **a constant** in your subclasses that stores information about the subclass. Let the constant that is common to the MyCar class and MyTruck class is **NUMER_OF_DOORS.**
- Your superclass should be able to keep track of the number of objects created that inherit from the superclass. Create a method to print out the value of this class variable as well.
    a. Let that variable be **number_of_vehicles**
    b. You need to define **instance variables; year, model, color, and current_speed**
    c. **Current_speed** should be set to zero at the time of creation of the object. d. Other variables can be initialized at the time of creation of the object e. You need to define a function **speed_up(number)** that will increase the speed of the vehicle by the amount "number"
    f. You need to define a function **brake(number)** that will decrease the speed of the vehicle by "number"
    g. You need to define a function **current_speed** that will print the current speed
    h. You need to define a function **shut_down** which will set the speed to zero with the message "Let's park the car! or Truck!". Car or Truck work should depend if the instance is a car or a truck.
    i. You need to define a function **spray_paint (given_color)** that will change the color to the given_color.
    j. You will define a function **gas_mileage(number1, number2)** that will calculate the millage. Round the result to the nearest integer.
- Create a **module Towable** that you can mix into your subclasses a. The module has a function **can_tow?(pounds)** which will check whether the pounds variable is less than 2000 or not.

- Design it using the Object Oriented Programming Approach. Functions and variables common to subclasses should be moved to the superclass. We will check this. Marks will be deducted if this approach is violated.

Example Output:

```
puts lumina = MyCar.new(1997, 'chevy lumina', 'white')

puts lumina.speed_up(20)

puts lumina.current_speed

puts lumina.speed_up(20)

puts lumina.current_speed

puts lumina.to_s

puts lumina.brake(20)

puts lumina.current_speed

puts lumina.brake(20)

puts lumina.current_speed

puts lumina.shut_down

puts MyCar.gas_mileage(13, 351)

puts lumina.spray_paint("red")

puts ram = MyTruck.new(1990, 'GMC', "black")

puts ram.can_tow?(1000)

puts lumina.can_tow?(3000)
```

>>> Output

```
My car is a white, 1997, chevy lumina!
You push the gas and accelerate 20 mph.
You are now going 20 mph.
You push the gas and accelerate 20 mph.
You are now going 40 mph.
My car is a white, 1997, chevy lumina!
You push the brake and decelerate 20 mph.
You are now going 20 mph.
You push the brake and decelerate 20 mph.
You are now going 0 mph.
Let's park the car!
27 miles per gallon of gas
Your new red paint job looks great!
My truck is a black, 1990, GMC!
true
false
```

Submission Guidelines

The file name should be vehicle.rb.

The file should be organized as follows:

```
class Vehicle
  …
end

class MyCar < Vehicle
  …
```

```
end

class MyTruck < Vehicle
  …
end

# Your own test cases start here
lumina = MyCar.new(1997, 'chevy lumina', 'white')
puts lumina.speed_up(20) # You push the gas and accelerate 20 mph.
puts lumina.current_speed # You are now going 20 mph.
puts lumina.speed_up(20) # You push the gas and accelerate 20 mph.
puts lumina.current_speed # You are now going 40 mph.
puts lumina.to_s # My mycar is a white, 1997, chevy lumina!
puts lumina.brake(20) # You push the brake and decelerate 20 mph.
puts lumina.current_speed # You are now going 20 mph.
puts lumina.brake(20) # You push the brake and decelerate 20 mph.
puts lumina.current_speed # You are now going 0 mph.
puts lumina.shut_down # Let's park the mycar!
```