Assignment #2
CS 3060 Programming Languages, Fall 2022
Instructor: S. Roy

Ruby #2

Due Date: Sep 16. 11:59 pm

Total points: 60 points

Directions: Using the source provided via Gitlab `https://gitlab.com/sanroy/fa22-cs3060-hw/`, complete the assignment below. The process for completing this assignment should be as follows:

1. You already forked the Repository “sanroy/fa22-cs3060-hw” to a repository “yourId/fa22-cs3060-hw” under your username. If not, do it now. Make sure that your repository is “private”; if you fail to do so, you will lose 20% of the points.
2. Get a copy of hw2 folder in “sanroy/fa22-cs3060-hw” repository as a hw2 folder in your repository “yourId/fa22-cs3060-hw”
3. Complete the assignment, committing changes to git. Each task code should be in a separate ruby file. As an example, task1.rb for Task 1.
4. Push all commits to your Gitlab repository
5. If you have done yet done so, **add** TA Akhil (gitlab id **ayerrab**) and Roy (gitlab id **sanroy**) as a member (in ‘Developer’ mode) of your Gitlab repository

Tasks:

1. **(12 points) Task #1:** Write a program that takes a filename f as input, and checks whether file f contains any word which starts with a capital letter (i.e., A-Z) and ends with a small letter (i.e., a-z) and contains a vowel (i.e., a,e,i,o, or u). As an example, "Apple", "Pen2Pen", or so can be such a word. Your program needs to print each line (in file f) that contains a word conforming the above pattern. Your program should also print the line numbers of the matching lines. *Writing readme carries 1 point.*
   Example run: ruby task1.rb file-to-search-in
   An example output is as follows.
   *Line 3: sdf **Apple** bjAbc bjb*
   *Line 7: Bgjh 12bask 13hj **Pen2Pen***
   Note: 1. Your project repository needs to include a sample file-to-search-in. 2. Do NOT use any library’s grep function. You implement it on your own, which may take only few lines of code. You can utilize *regular expression.*
2. **(12 points) Task #2:** Write a function that takes an array A of integers as the input and does the following: (a) Uses *each* method to print the last digit of each integer in A, (b) Uses *select* method to find all the integers (in A) whose lengths are less then 4, (c) Uses *map* method to build a new array with the length of the integers of A, and (d) Uses *inject* method to find the sum of the length of all integers of A. To test the function, build an array A of 12 random integers, and pass A into the function as a parameter. *Writing readme carries 1 point.*
3. **(10 points) Task #3:**. Function3A calculates the Fibonacci Series ($F_n = F_{n-1} + F_{n-2}$) iteratively. Calculate all Fibonacci numbers (F_n) for $n = 1$ to 38. Function3B calculates the same Fibonacci Series but using the recursion technique. After implementing the above two functions, compare the computation time (for doing the above calculation) using Ruby’s benchmark library. *Writing readme carries 1 point.* **Hint**: if necessary, Ruby lecture slides (ppt) can help you write the iterative version.
4. **(14 points) Task #4:** The Tree class presented in the textbook (Day 2) chapter is interesting, but it does not allow you to specify a new tree with a clean user interface. (a) Update the constructor (and other methods if necessary) to accomodate the creation of a tree using a Hash. The constructor should accept a nested

structure of Hashes. You should be able to specify a tree as below. To test your functionality, you should traverse (by calling visit_all method on the tree, i.e., the root node) this entire tree and print out the node names. (b) Write more code which counts the total number of nodes on the tree and also counts the total number of leaves on the tree.

```
ruby_tree = Tree.new({
    'ggp' => {
        'gp1' =>
            {'p1' => {'child1' => {}, 'child2' => {}},
             'p2' => {'child3' => {}}
            },
        'gp2' =>
            {'p3' => {'child4' => {}},
             'p4' => {'child5' => {}, 'child6' => {}}
            },
        'gp3' =>
            {'p5' => {'child7' => {}},
             'p6' => {'child8' => {}}
            }
    }
})
```

Writing readme carries 1 point.
Hint: for part (a) you may mainly modify the *initialize* function of the Tree class.

5. **(12 points) Task #5:** In this task, your code creates a random list of 50 shape objects, then traverses the list from start to end, and computes the total area of the shape objects. First you need to implement the class hierarchy diagram of the shape types, which is attached. Shape is an abstract class which has only a "color" attribute whereas (regular) Octagon class and (regular) Hexagon class are concrete children of Shape class, and they have more attributes and constructors. Note that you do not know beforehand the order of the shape objects (i.e. Octagons and Hexagons) created in the random list, e.g., you do not know beforehand whether the 1st item is Octagon or Hexagon. *Writing readme carries 1 point.*
Note. When you traverse the list to calculate the total area, you call the area() function of each shape object (without considering whether it is Ocagon or Hexagon). That means, you will use the concept of polymorphism.
Hint: While building the list of shape objects, use rand(2) to generate a random number 0 or 1; if 0, then you may add a Octagon object, else add a Hexagon object to the list.