# CSE/ISE 337: Scripting Languages
# Stony Brook University
# Programming Assignment #2
# Fall 2020
# Assignment Due: 30th October, Monday, by 11:59 PM

# (EST)

### Learning Outcomes

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

- Design and implement scripts in Ruby
- Extend existing Ruby classes with your own functionality
- Design and implement a detailed 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
- **There are 4 files provided with the assignment (V.Imp -> Don't change the function definition) You can add your own functions to make the code more modular but don't change the existing functions.**
  - First file array.rb
  - Second file rgrep.rb
  - Third file contains_virus.rb and Bonus file contains_virus_bonus.rb
- You can use the test cases provided in the files or create your own test cases.
- You can utilize but not limited to the class definition provided in the files. You can create new classes which suit your needs.
- 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 own test cases to thoroughly test your code.
- All code must be written in Ruby.

**Problem 1 (30 points)**

**T**he *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 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 bound.

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

b = ["cat","bat","mat","sat"]
b[-1] = "sat"
b[5] = '\0'
b.map(2..10) { |x| x[0].upcase + x[1,x.length] } = ["Mat", "Sat"] b.map(2..4) { |x| x[0].upcase + x[1,x.length] } = ["Mat", "Sat"] b.map(-3..-1) { |x| x[0].upcase + x[1,x.length] } = ["Bat", "Mat", “Sat”] 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**

**Problem 2 (40 points)**

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. 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.

• -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.

Usage

A user of rgrep should provide a file name (fully qualified path if not in 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 other 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**

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 (30 points)**

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 isInfected

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

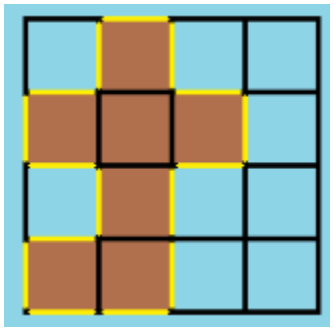
If the value isInfected[i][j] == 0 represents uninfected cells, and if isInfected[i][j] == 1 represents cells contaminated with the virus.

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 region. So, our Goal is to find the number of walls built to contain the virus. Please note: In this question, the constraint is there is only one area where the virus will be. 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 places 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. The cell with row 1 and col 1, even though it contains virus, we don't have to make any walls for it. So we don't include this in the final answer.**

**Bonus Question (10 Marks)**

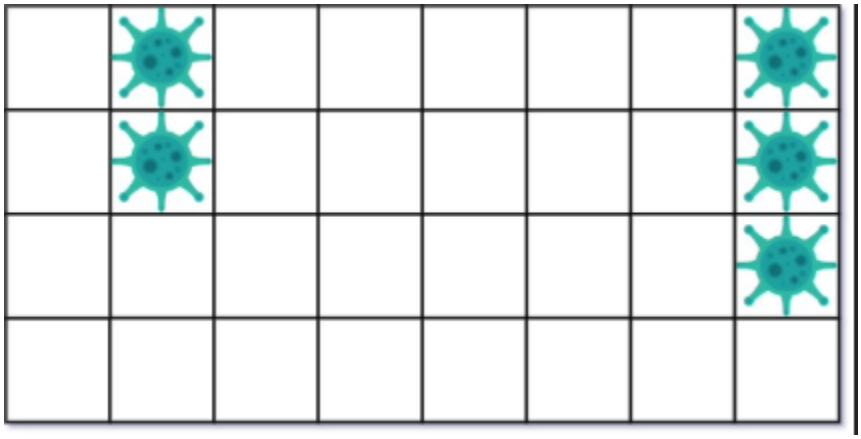
Continuing to the previous question, Every night, the virus spreads to all neighboring cells in all four directions(This does not mean diagonally, the virus only spreads in left, right, up and down) unless blocked by a wall.

Resources are limited. Each day, you can install walls around only one region (i.e., the affected area (continuous block of infected cells) that threatens the most uninfected cells the following night). There will never be a tie.

Return the number of walls used to quarantine all the infected regions. If the world becomes fully infected, return the number of walls used. Please Note: In this the virus can have different regions as well, they will not be placed at a single location as seen on the sample test case 1.

**Sample Test Case 1 ->**

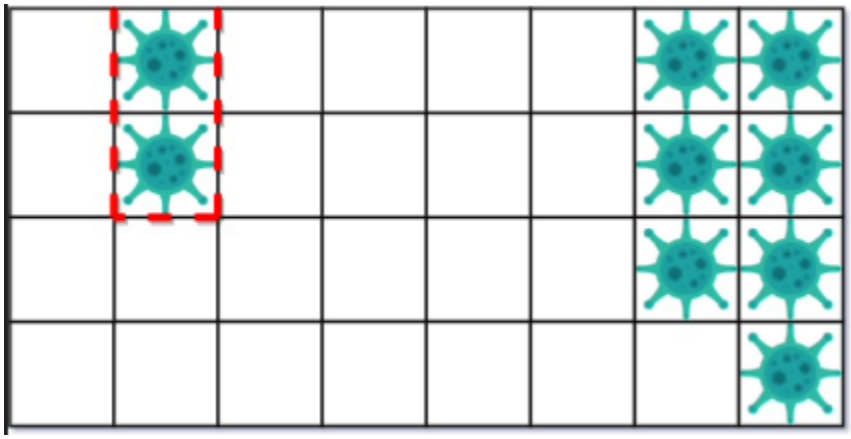
Input: isInfected = [[0,1,0,0,0,0,0,1],[0,1,0,0,0,0,0,1],[0,0,0,0,0,0,0,1],[0,0,0,0,0,0,0,0]]



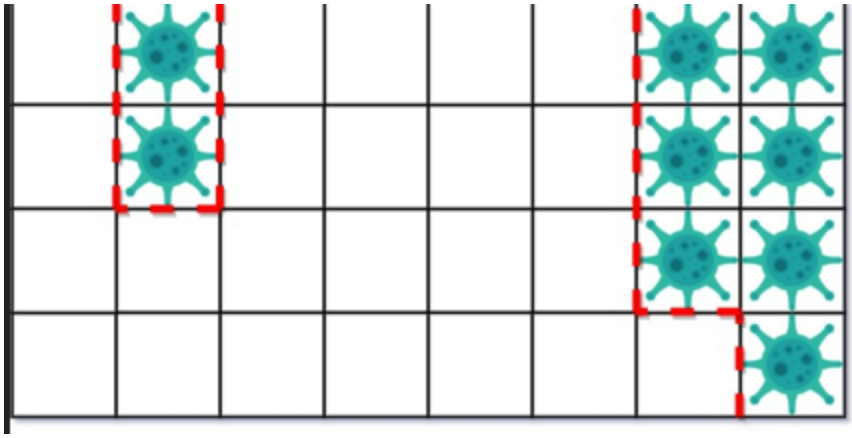
**Output: 10**

Explanation: There are 2 contaminated regions.

On the first day, add 5 walls to quarantine the viral region on the left. The board after the virus spreads is:

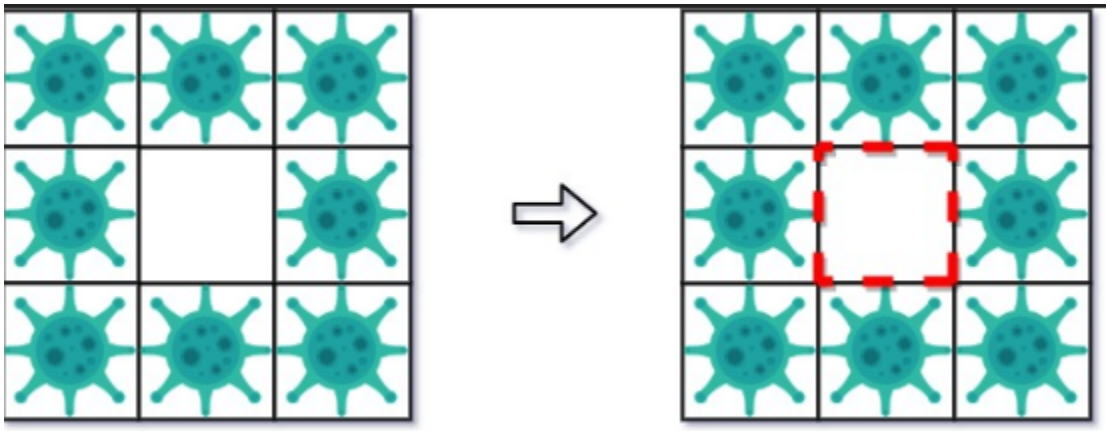


On the second day, add 5 walls to quarantine the viral region on the right. The virus is fully contained.



**Sample Test case 2 ->**

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



**Output: 4**

Explanation: Even though there is only one cell saved, there are 4 walls built.

Notice that walls are only built on the shared boundary of two different cells.