#3 - Polynomials

You just started working for CoolNewCompany which is developing mathematics related software. Since you are new to the team, your boss gives you an easy task to test your abilities. Write a class that pretty-prints polynomials, following some simple rules:

- if a coefficient is 1, it doesn't get printed
- if a coefficient is negative, you have to display something like "- 2x^3", not "+ -2x^3"
- if a coefficient is 0, nothing gets added to the outputt
- for x^1 the ^1 part gets omitted
- x^0 == 1, so we don't need to display it

Here's a couple of usage examples:

```
puts Polynomial.new([-3,-4,1,0,6]) # => -3x^4-4x^3+x^2+6
puts Polynomial.new([1,0,2]) # => x^2+2
```

Don't concern yourself too much with error handling, but if somebody tries to create a polynomial with less than 2 elements, your program has to raise an **ArgumentError** with the message "Need at least 2 coefficients."

Please check the provided unit tests for more examples and make sure to use them for verifying your solution!

Requirements: This has to be a pure Ruby script, using only the Ruby Standard Libraries (meaning, no external Gems). You **do not** need to build a gem for this. Pure Ruby code is all that is needed.

#4 - Mazes

Mazes are known to have challenged humans from as far back as the 5th century BC. There are many types of maze, but typically you need to find your way from a start point to an end point.

In this Ruby challenge, you will need to develop a class that can be used to solve mazes. Mazes will be provided as a string showing a graphical representation of the maze's layout. Spaces are navigable, while # (pound) symbols are used to denote walls. In this challenge the letter "A" is used to mark the start point, and "B" the end point. Here's an example of a maze contained within a string:

```
MAZE1 = %{####################################
# #   #     #A        #     #       #
# # # # # # ####### # ### # ####### #
# # #   # #         #     # #       #
# ##### # ################# # #######
#     # #       #   #     # #   #   #
##### ##### ### ### # ### # # # # # #
#   #     #   # #   #  B# # # #   # #
# # ##### ##### # # ### # # ####### #
# #     # #   # # #   # # # #       #
# ### ### # # # # ##### # # # ##### #
#   #       #   #       #     #     #
####################################}
```

The prior maze would be loaded into a Maze object like so:

```
Maze.new(MAZE1)
```

The Challenge

There are two parts to the challenge: you can choose to do one or both, depending on your skill level or how much time you have available.

1. Implement a Maze#solvable? method that returns true/false depending on whether it's possible to navigate the maze from point A to point B.
2. Implement a Maze#steps method that returns an integer of the least number of "steps" one would have to take within the maze to get from point A to point B. "Steps" can only be taken up, down, left or right. No diagonals.

There are a number of ways to "solve" mazes but there's a wide scope for you to be as straightforward or as clever as you like with this challenge (tip: I'd love to see some clever/silly solutions!). Your "solvable?" and "steps" methods could share algorithms or you might come up with alternate ways to be more efficient in each case. Good luck!

Note: Use the test suite to ensure your code interfaces in the right way. The test suite demonstrates how your class will be called and used.

The Test Suite

Your Maze class should be in maze.rb for the following test suite to work 😀 . Also do `gem install 'minitest'`

```
require 'minitest/autorun'
require 'maze'

MAZE1 = %{####################################
# #   #     #A        #     #       #
# # # # # # ####### # ### # ####### #
# # #   # #         #     # #       #
# ##### # ################# # #######
#     # #       #   #     # #   #   #
```

```
##### ##### ### ### # ### # # # # # # #
#   #     #   # #   #  B# # # #   # #
# # ##### ##### # # ### # # ####### #
# #     # #   # # #   # # # #       #
# ### ### # # # # ##### # # # ##### #
#   #       #   #       #     #     #
#####################################}
# Maze 1 should SUCCEED

MAZE2 = %{#####################################
# #       #             #     #     #
# ### ### # ########### ### # ##### #
# #   # #   #   #   #   #   #       #
# # ###A##### # # # # ### ###########
#   #   #     #   # # #   #         #
####### # ### ####### # ### ####### #
#       # #   #       # #       #   #
# ####### # # # ####### # ##### # # #
#       # # # #   #       #   # # # #
# ##### # # ##### ######### # ### # #
#     #   #                 #     #B#
#####################################}
# Maze 2 should SUCCEED

MAZE3 = %{#####################################
# #   #           #                 #
# ### # ####### # # # ############# #
#   #   #     # #   # #     #     # #
### ##### ### ####### # ##### ### # #
# #       # #  A  #   #       #   # #
# ######### ##### # ####### ### ### #
#               ###       # # # #   #
# ### ### ####### ####### # # # # ###
# # # #   #     #B#   #   # # #   # #
# # # ##### ### # # # # ### # ##### #
#   #         #     #   #           #
#####################################}
# Maze 3 should FAIL


class MazeTest < Minitest::Test
  def test_good_mazes
    assert_equal true, Maze.new(MAZE1).solvable?
    assert_equal true, Maze.new(MAZE2).solvable?
  end

  def test_bad_mazes
    assert_equal false, Maze.new(MAZE3).solvable?
  end

  def test_maze_steps
    assert_equal 44, Maze.new(MAZE1).steps
    assert_equal 75, Maze.new(MAZE2).steps
    assert_equal 0, Maze.new(MAZE3).steps
  end
end
```

Requirements(as usual)

This has to be a pure Ruby script, using only the Ruby Standard Libraries (meaning, no external Gems). You **do not** need to build a gem for this. Pure Ruby code is all that is needed.

#6 - Business Hours

Chunky Bacon Begone is a dry-cleaning company known for its speedy service. It guarantees to dry-clean anything within two business hours or less. The problem is, when the customer drops off the clothes, he needs to know what time they are guaranteed to be done.

It is your job to write a Ruby script which will determine the guaranteed time given a business hour schedule. You must create a class called **BusinessHours** which allows one to define the opening and closing time for each day. It should provide the following interface:

```
hours = BusinessHours.new("9:00 AM", "3:00 PM")
hours.update :fri, "10:00 AM", "5:00 PM"
hours.update "Dec 24, 2010", "8:00 AM", "1:00 PM"
hours.closed :sun, :wed, "Dec 25, 2010"
```

The **update** method should change the opening and closing time for a given day. The **closed** method should specify which days the shop is not open. Notice days can either be a symbol for week days or a string for specific dates. Any given day can only have one opening time and one closing time — there are no off-hours in the middle of the day.

A method called **calculate_deadline** should determine the resulting business time given a time interval (in seconds) along with a starting time (as a string). The returned object should be an instance of **Time**. Here are some examples:

```
hours.calculate_deadline(2*60*60, "Jun 7, 2010 9:10 AM") # => Mon Jun 07 11:10:00 2010
hours.calculate_deadline(15*60, "Jun 8, 2010 2:48 PM") # => Thu Jun 10 09:03:00 2010
hours.calculate_deadline(7*60*60, "Dec 24, 2010 6:45 AM") # => Mon Dec 27 11:00:00 2010
```

In the first example the time interval is 2 hours (7,200 seconds). Since the 2 hours fall within business hours the day does not change and the interval is simply added to the starting time.

In the second line an interval of 15 minutes (900 seconds) is used. The starting time is 12 minutes before closing time which leaves 3 minutes remaining to be added to the next business day. The next day is Wednesday and therefore closed, so the resulting time is 3 minutes after opening on the following day.

The last example is 7 hours (25200 seconds) which starts before opening on Dec 24th. There are only 5 business hours on Dec 24th which leaves 2 hours remaining for the next business day. The next two days are closed (Dec 25th and Sunday) therefore the deadline is not until 2 hours after opening on Dec 27th.

Tip: Use **Time.parse** to generate a Time from a string. You may need to **require "time"** in order to do this.

Requirements: This has to be a pure Ruby script, using only the Ruby Standard Libraries (meaning, no external Gems, libraries). You do not need to build a gem for this. Pure Ruby code is all that is needed.

We will mostly be judging by the beauty of your code as long as it satisfies the test cases.

#7 - Cycle Tracks

https://drive.google.com/open?id=0B_lWyPvhnJTVUnFyam5lZFlsUHc&authuser=1