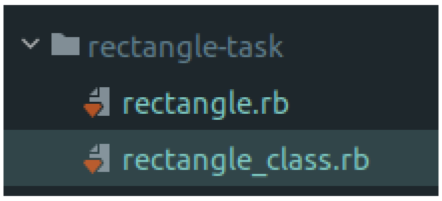
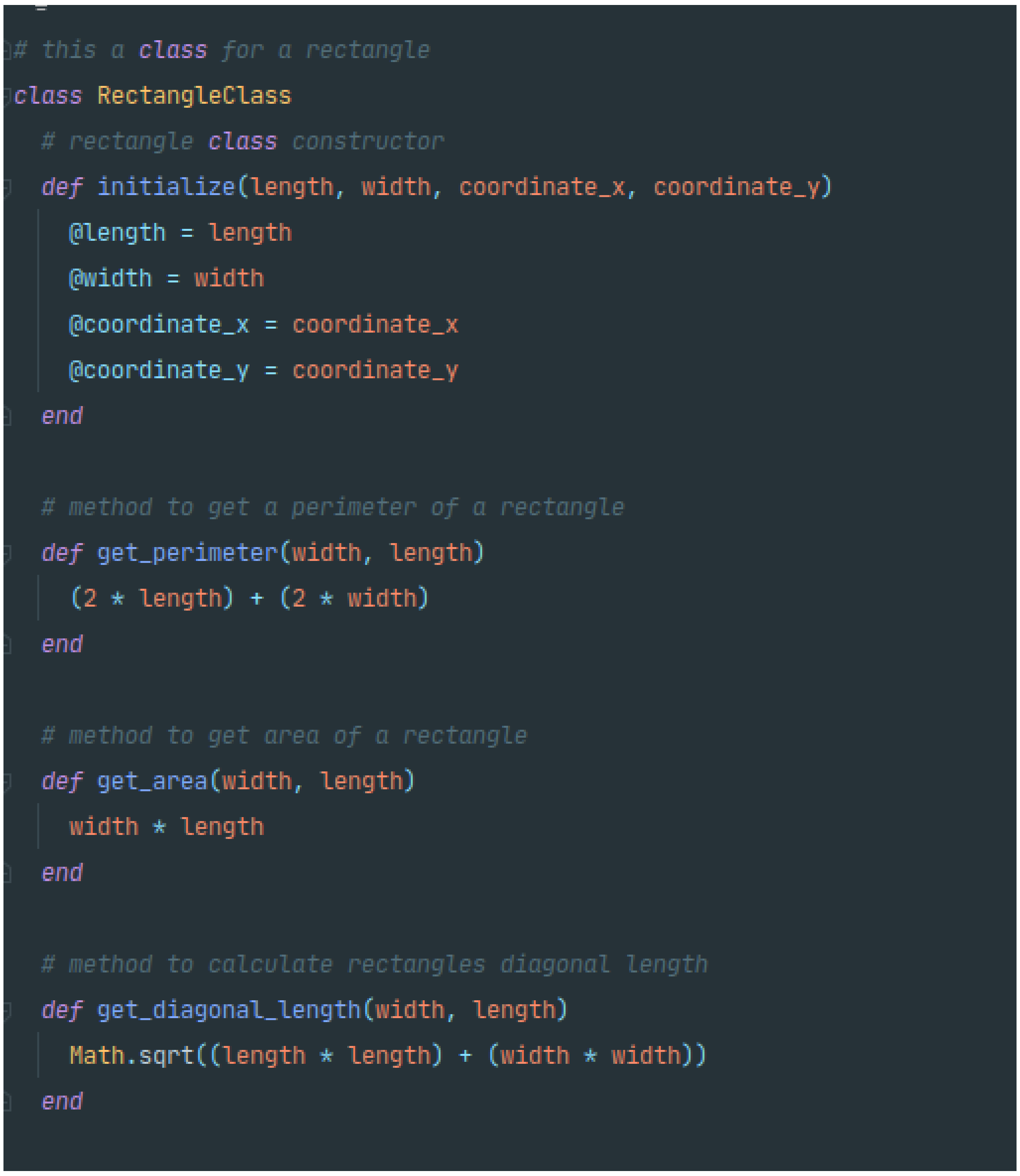
TASK3 – rectangle-task

The only difference between this task and the last one (TASK2) is that this task has to have OOP principles. So this task has rectangle and rectangle class files.

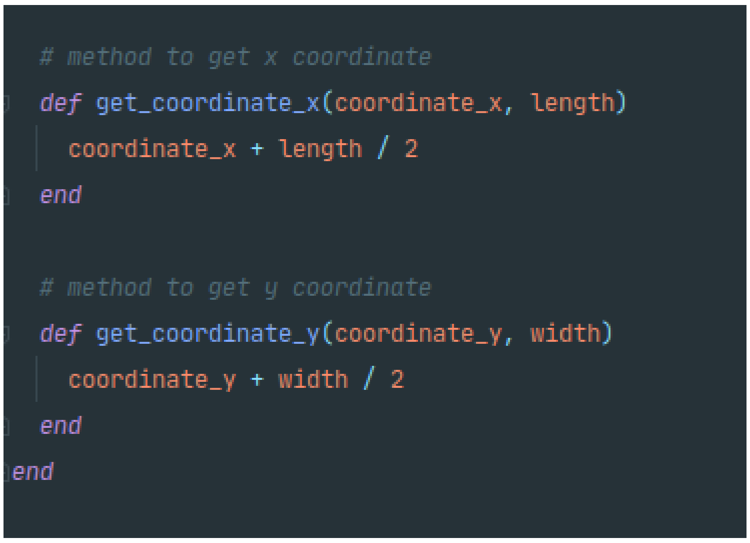


```
rectangle-task
    rectangle.rb
    rectangle_class.rb
```

In *rectangle_class.rb* I've written a rectangle class, constructor for it to be called and methods.

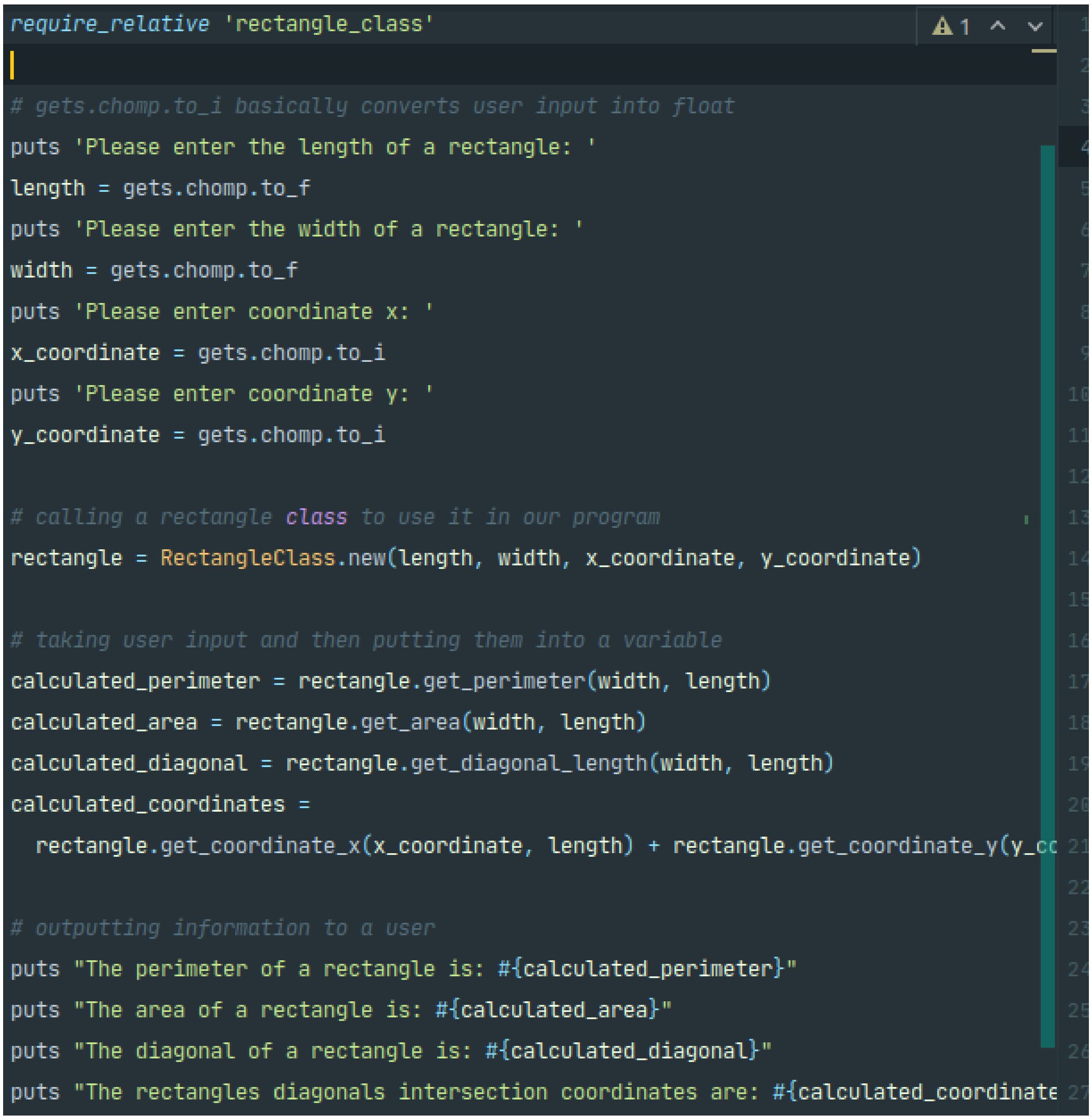


```ruby
# this a class for a rectangle
class RectangleClass
  # rectangle class constructor
  def initialize(length, width, coordinate_x, coordinate_y)
    @length = length
    @width = width
    @coordinate_x = coordinate_x
    @coordinate_y = coordinate_y
  end

  # method to get a perimeter of a rectangle
  def get_perimeter(width, length)
    (2 * length) + (2 * width)
  end

  # method to get area of a rectangle
  def get_area(width, length)
    width * length
  end

  # method to calculate rectangles diagonal length
  def get_diagonal_length(width, length)
    Math.sqrt((length * length) + (width * width))
  end
```



```ruby
  # method to get x coordinate
  def get_coordinate_x(coordinate_x, length)
    coordinate_x + length / 2
  end

  # method to get y coordinate
  def get_coordinate_y(coordinate_y, width)
    coordinate_y + width / 2
  end
end
```

And this below is a *rectangle.rb* where we call our class and perform some calculations.



```ruby
require_relative 'rectangle_class'

# gets.chomp.to_i basically converts user input into float
puts 'Please enter the length of a rectangle: '
length = gets.chomp.to_f
puts 'Please enter the width of a rectangle: '
width = gets.chomp.to_f
puts 'Please enter coordinate x: '
x_coordinate = gets.chomp.to_i
puts 'Please enter coordinate y: '
y_coordinate = gets.chomp.to_i

# calling a rectangle class to use it in our program
rectangle = RectangleClass.new(length, width, x_coordinate, y_coordinate)

# taking user input and then putting them into a variable
calculated_perimeter = rectangle.get_perimeter(width, length)
calculated_area = rectangle.get_area(width, length)
calculated_diagonal = rectangle.get_diagonal_length(width, length)
calculated_coordinates =
  rectangle.get_coordinate_x(x_coordinate, length) + rectangle.get_coordinate_y(y_c

# outputting information to a user
puts "The perimeter of a rectangle is: #{calculated_perimeter}"
puts "The area of a rectangle is: #{calculated_area}"
puts "The diagonal of a rectangle is: #{calculated_diagonal}"
puts "The rectangles diagonals intersection coordinates are: #{calculated_coordinate
```

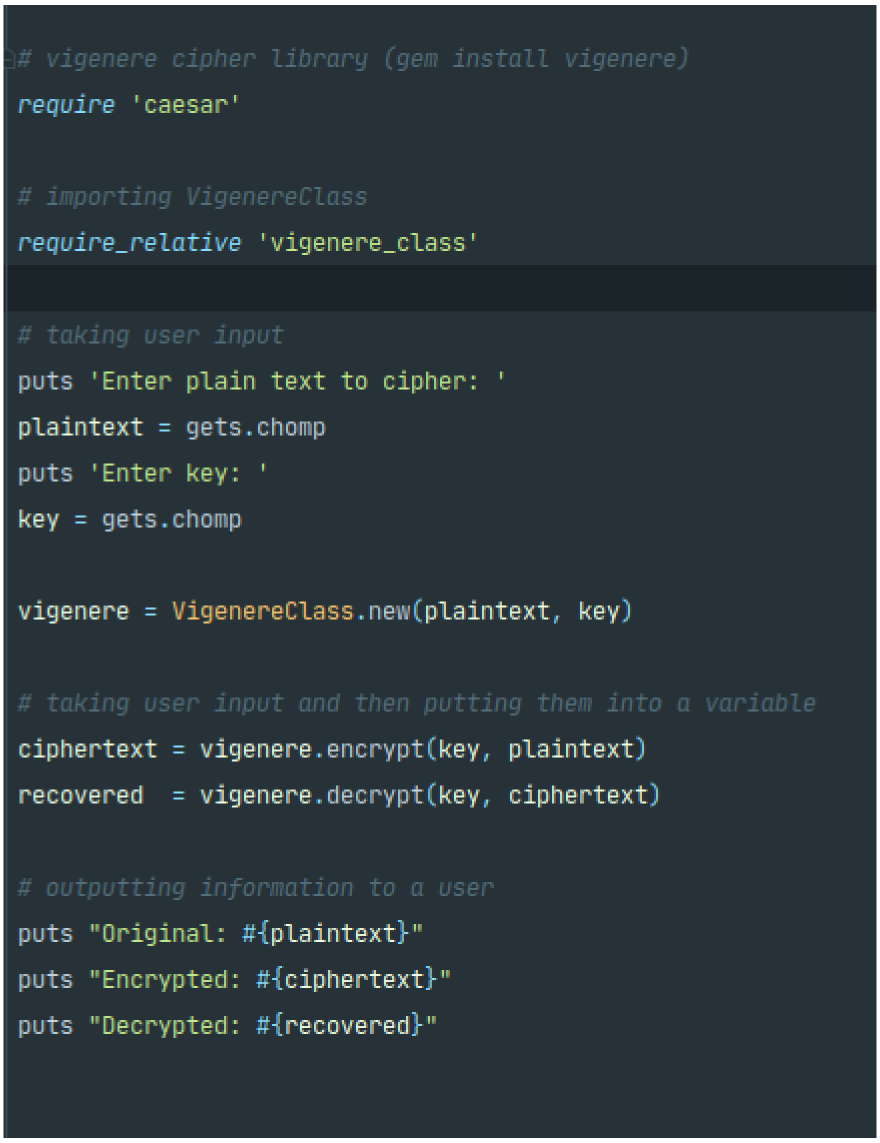
First we link our file that we’re going to import our class from. Then we get user inputs, we create an object of a rectangle and then we call our methods from the class.

Outputs are the same as in the previous task.

TASK3 – vigenere-cipher

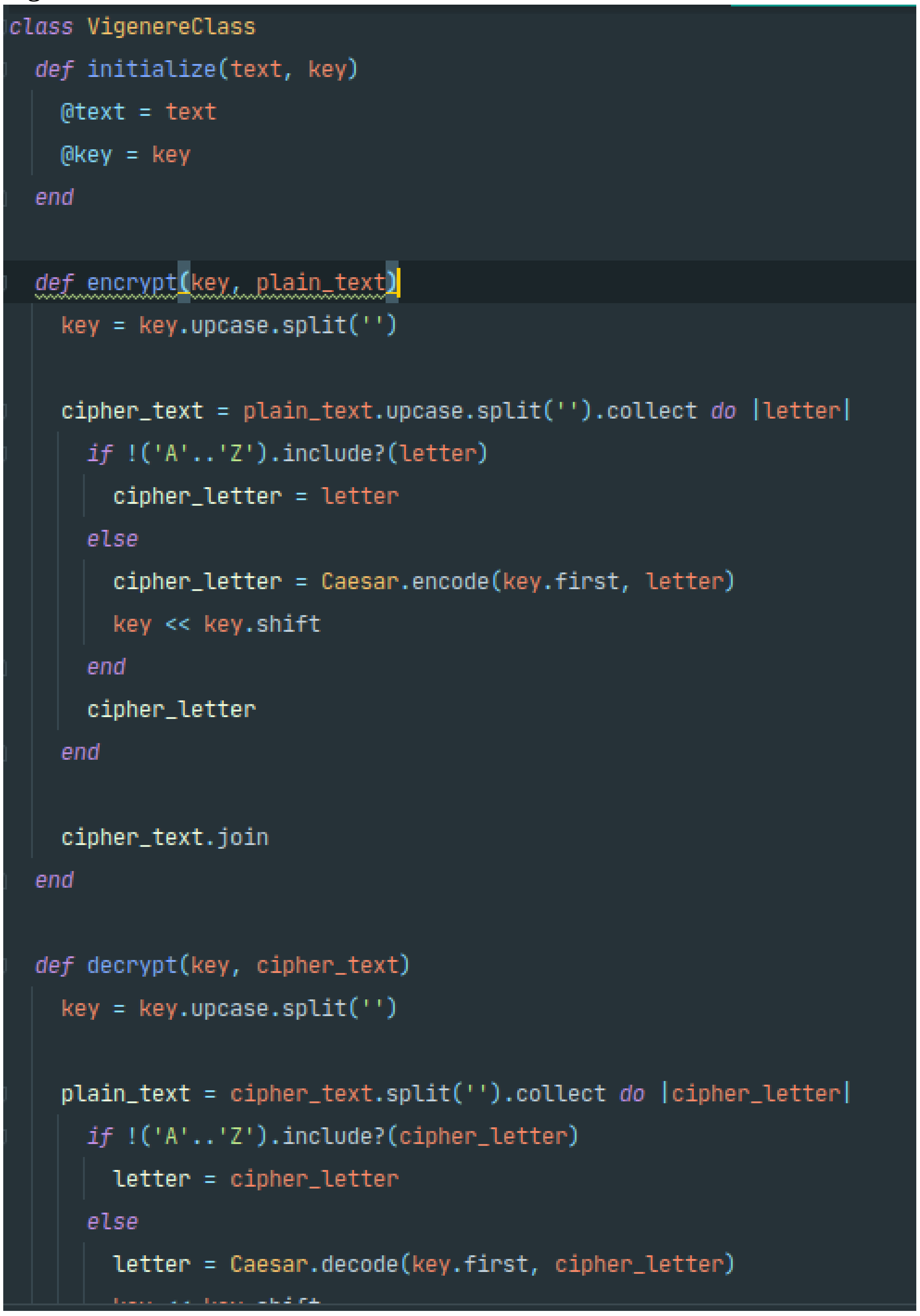
Same goes with vigenere. We link our class file, get user input, then we create an object and perform needed methods. Easy as that.

Vigenere.rb:

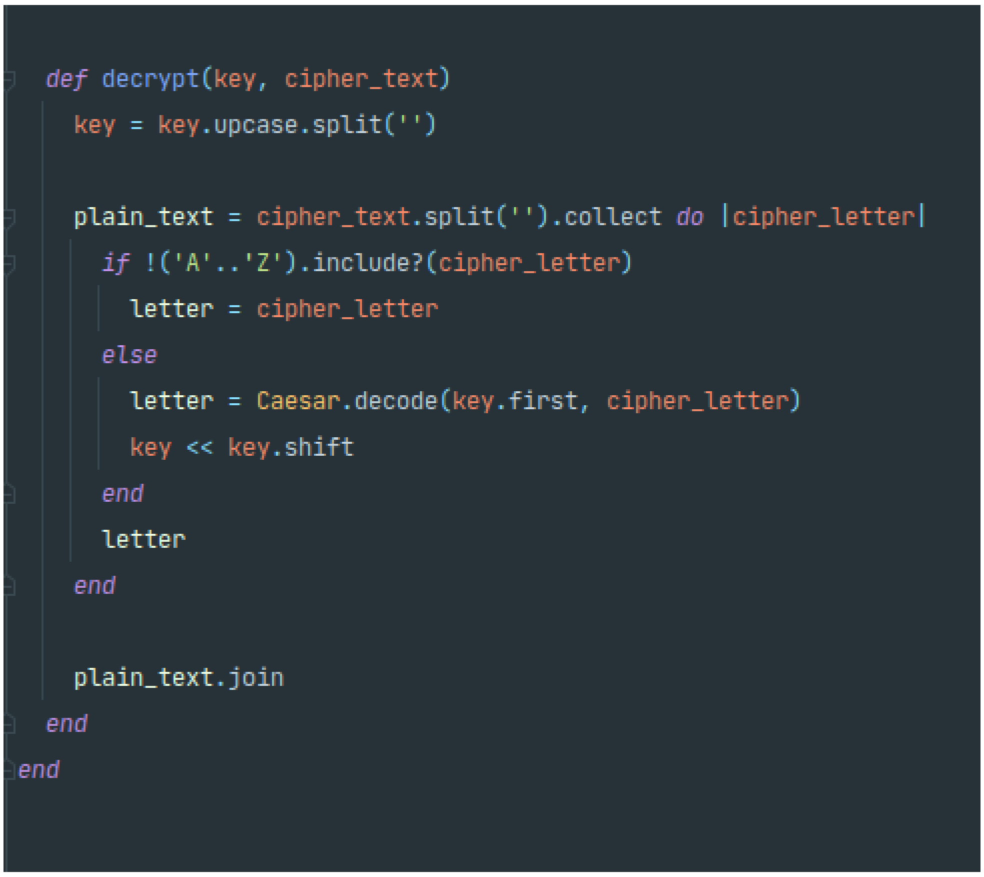


```ruby
# vigenere cipher library (gem install vigenere)
require 'caesar'

# importing VigenereClass
require_relative 'vigenere_class'

# taking user input
puts 'Enter plain text to cipher: '
plaintext = gets.chomp
puts 'Enter key: '
key = gets.chomp

vigenere = VigenereClass.new(plaintext, key)

# taking user input and then putting them into a variable
ciphertext = vigenere.encrypt(key, plaintext)
recovered  = vigenere.decrypt(key, ciphertext)

# outputting information to a user
puts "Original: #{plaintext}"
puts "Encrypted: #{ciphertext}"
puts "Decrypted: #{recovered}"
```

vigenere_class.rb:



```ruby
class VigenereClass
  def initialize(text, key)
    @text = text
    @key = key
  end

  def encrypt(key, plain_text)
    key = key.upcase.split('')

    cipher_text = plain_text.upcase.split('').collect do |letter|
      if !('A'..'Z').include?(letter)
        cipher_letter = letter
      else
        cipher_letter = Caesar.encode(key.first, letter)
        key << key.shift
      end
      cipher_letter
    end

    cipher_text.join
  end

  def decrypt(key, cipher_text)
    key = key.upcase.split('')

    plain_text = cipher_text.split('').collect do |cipher_letter|
      if !('A'..'Z').include?(cipher_letter)
        letter = cipher_letter
      else
        letter = Caesar.decode(key.first, cipher_letter)
```



```
  def decrypt(key, cipher_text)
    key = key.upcase.split('')

    plain_text = cipher_text.split('').collect do |cipher_letter|
      if !('A'..'Z').include?(cipher_letter)
        letter = cipher_letter
      else
        letter = Caesar.decode(key.first, cipher_letter)
        key << key.shift
      end
      letter
    end

    plain_text.join
  end
end
```

Outputs are the same as in the previous task.