**Ashish\_Quiz 2**

1. Ruby program when that accepts a date in a month and prints it out the appropriate suffix. For example, for 1 as input, print 1st, 2 as input 2nd, 3 as input 3rdetc

Ans :

2. Write your own ruby program using a case statement

Ans:

#!/usr/bin/env ruby

print "Enter your grade: "

grade = gets.chomp

case grade

when "A"

puts 'Well done!'

when "B"

puts 'Try harder!'

when "C"

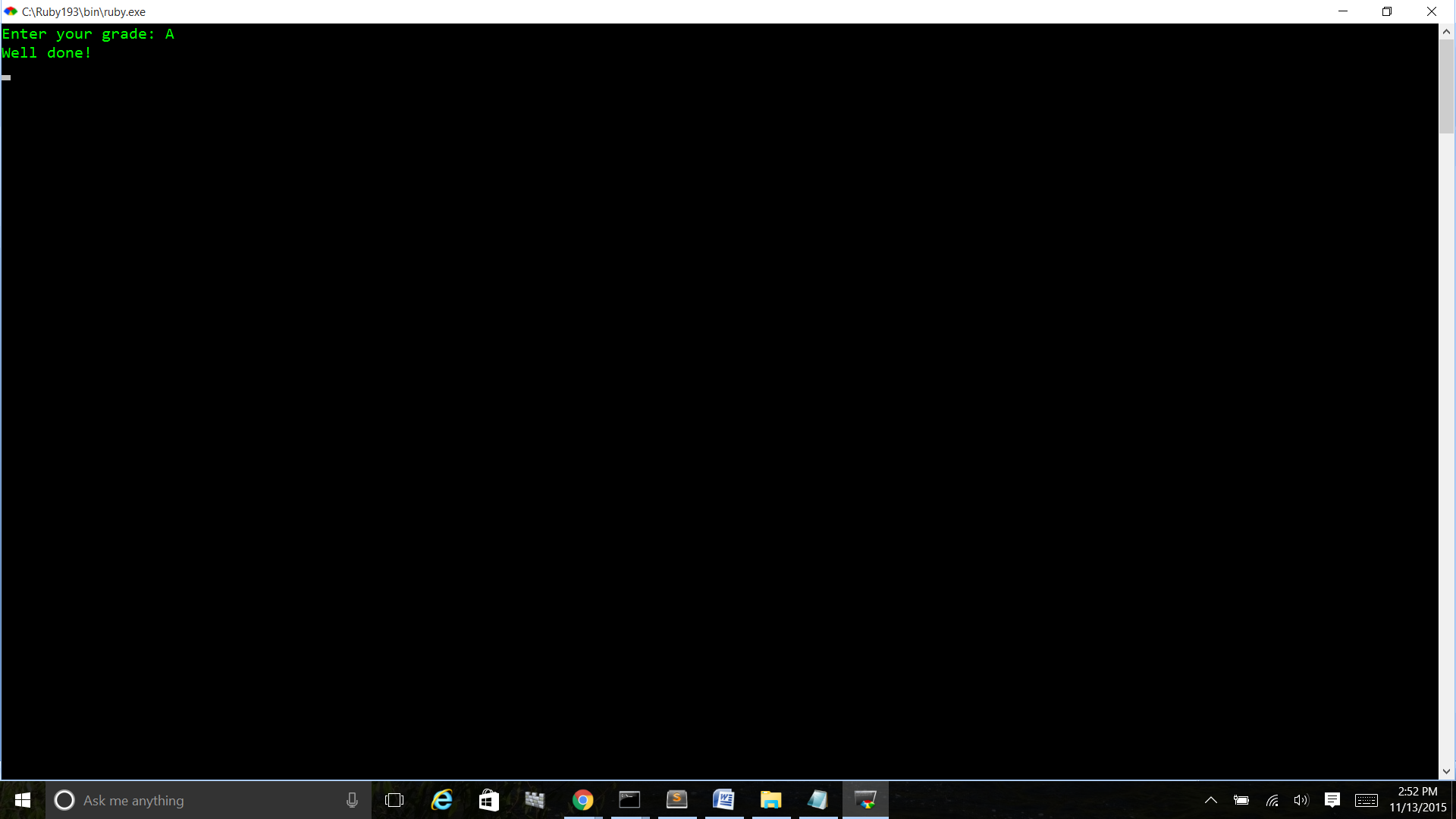
puts 'You need help!!!'

else

puts "You just making it up!"

end

gets

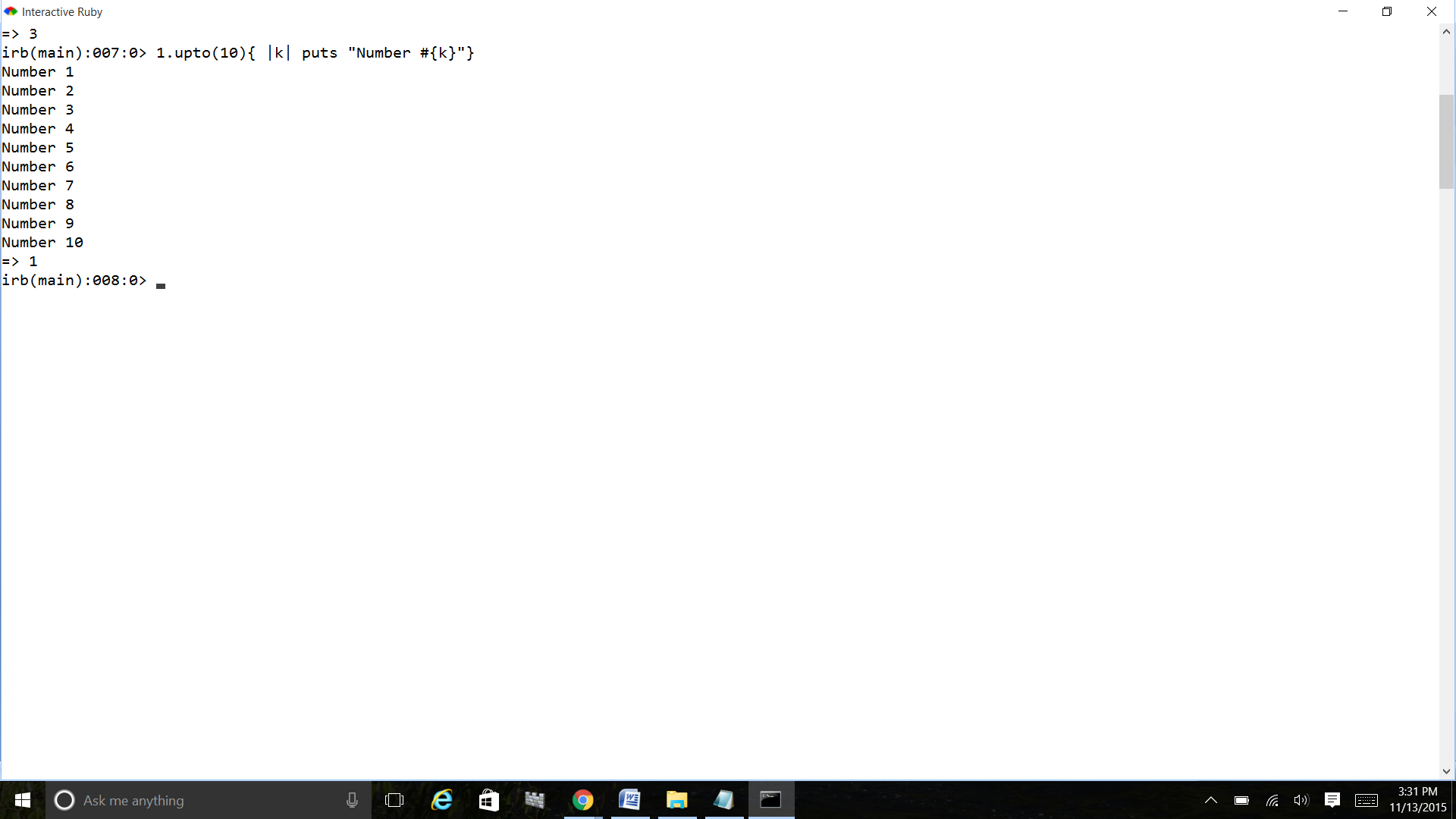


3. Ruby Program that iterates numbers with upto loop

Ans

#!/usr/bin/env ruby

1.upto(10){|k| puts "Number #{k}"}



4. Write a Ruby program to print numbers from 1 to 50 and also in reverse order

Ans:

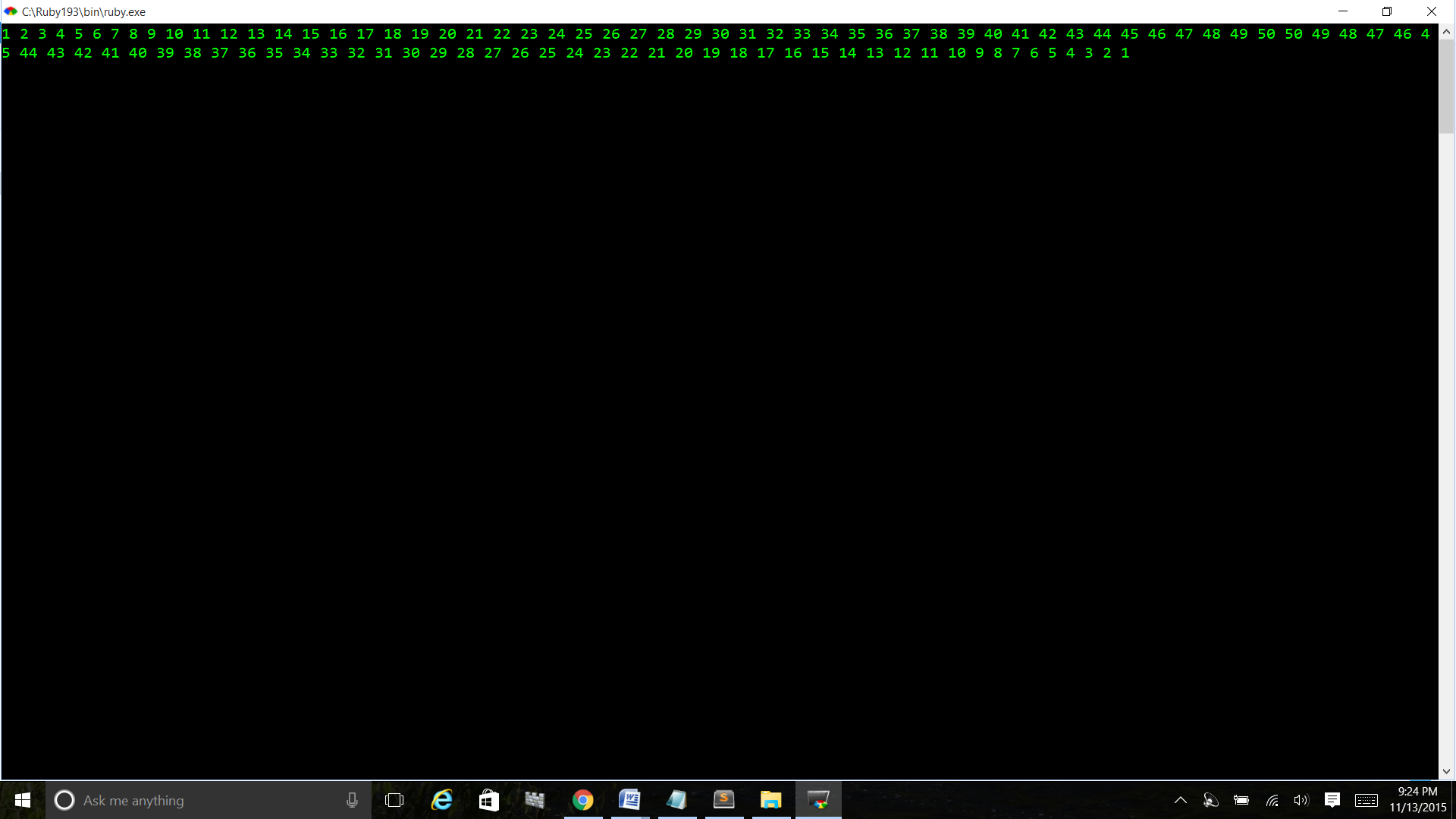
#!/usr/bin/env ruby

1.upto(50) { |i| print i, " " }

50.downto(1) { |i| print i, " " }

gets

**output**



5. Write your own Ruby program using loops and iterators. Explain the difference between loops, iterators and blocks

Ans: Loops in Ruby are used to execute the same block of code a specified number of times

#!/usr/bin/env ruby

# conditional\_loop.rb

i = 0

loop do

i += 2

puts "#{i}"

if i == 10

break

end

Both loops and iterators are used to repeat a chunk of code.

Loops are an ancient idea, they existed long before computers did.

Almost every programming language has some kind of loops.

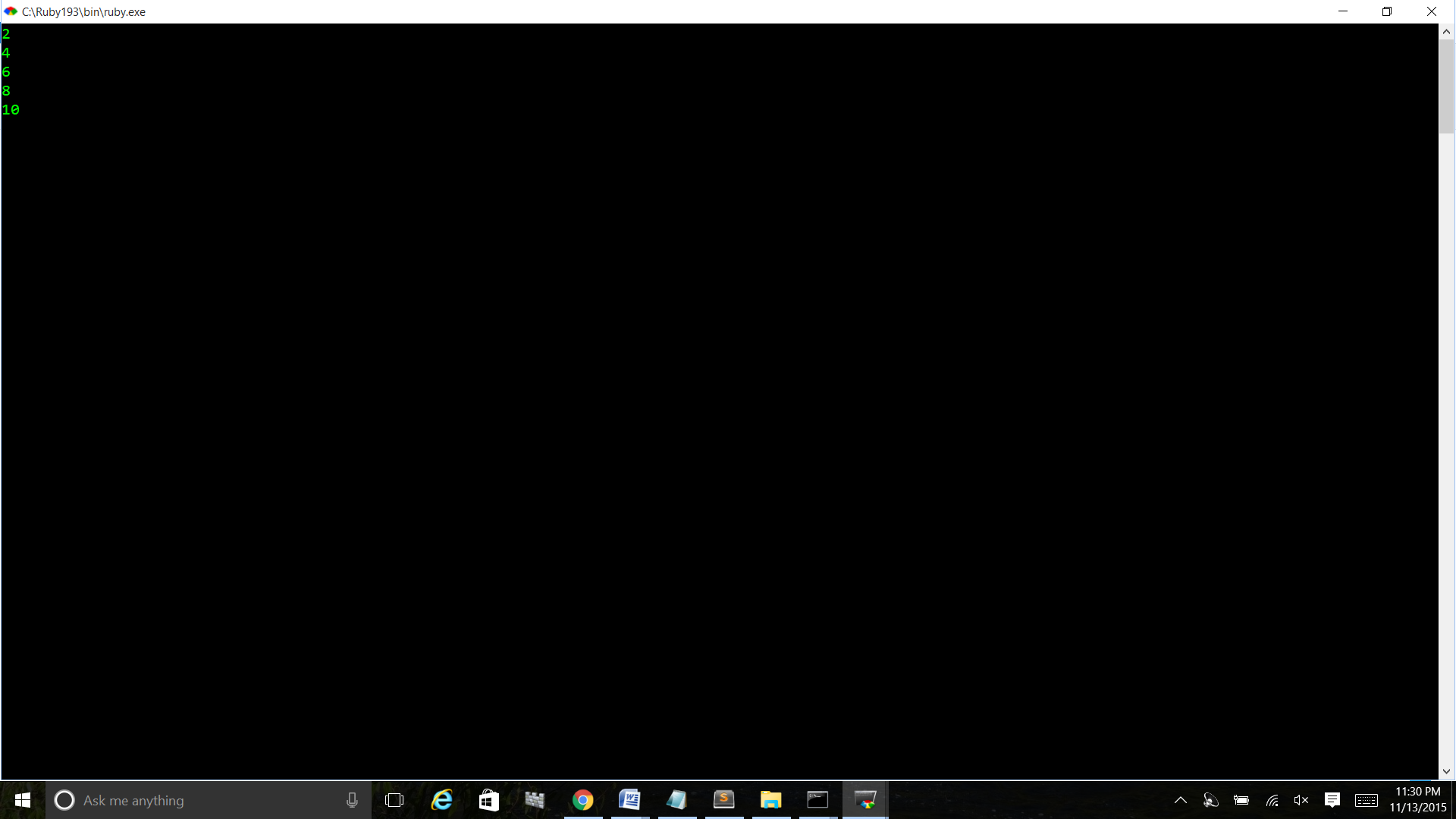
Iterators are relatively new, and they only exist in a few languages such as Ruby.

Ruby Code blocks (called closures in other languages) are definitely one of the coolest features of Ruby and are chunks of code between braces or between do- end that you can associate with method invocations, almost as if they were parameters.

A Ruby block is a way of grouping statements, and may appear only in the source adjacent to a method call; the block is written starting on the same line as the method call's last parameter (or the closing parenthesis of the parameter list).

The code in the block is not executed at the time it is encountered.

Instead, Ruby remembers the context in which the block appears (the local variables, the current object, and so on) and then enters the method.



Iterators:- The each iterator returns all the elements of an array or a hash.

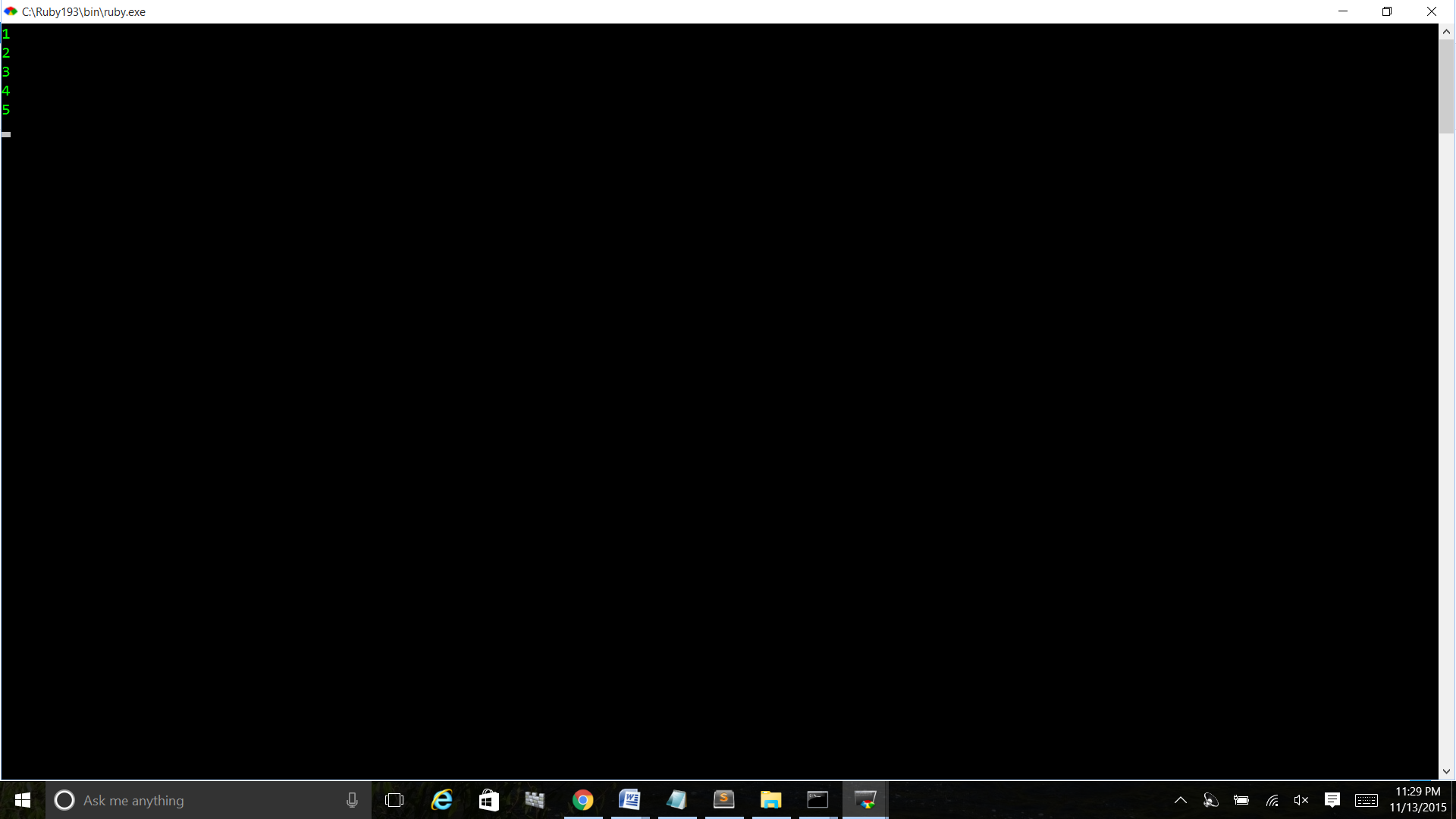
#!/usr/bin/ruby

ary = [1,2,3,4,5]

ary.each do |i|

puts i

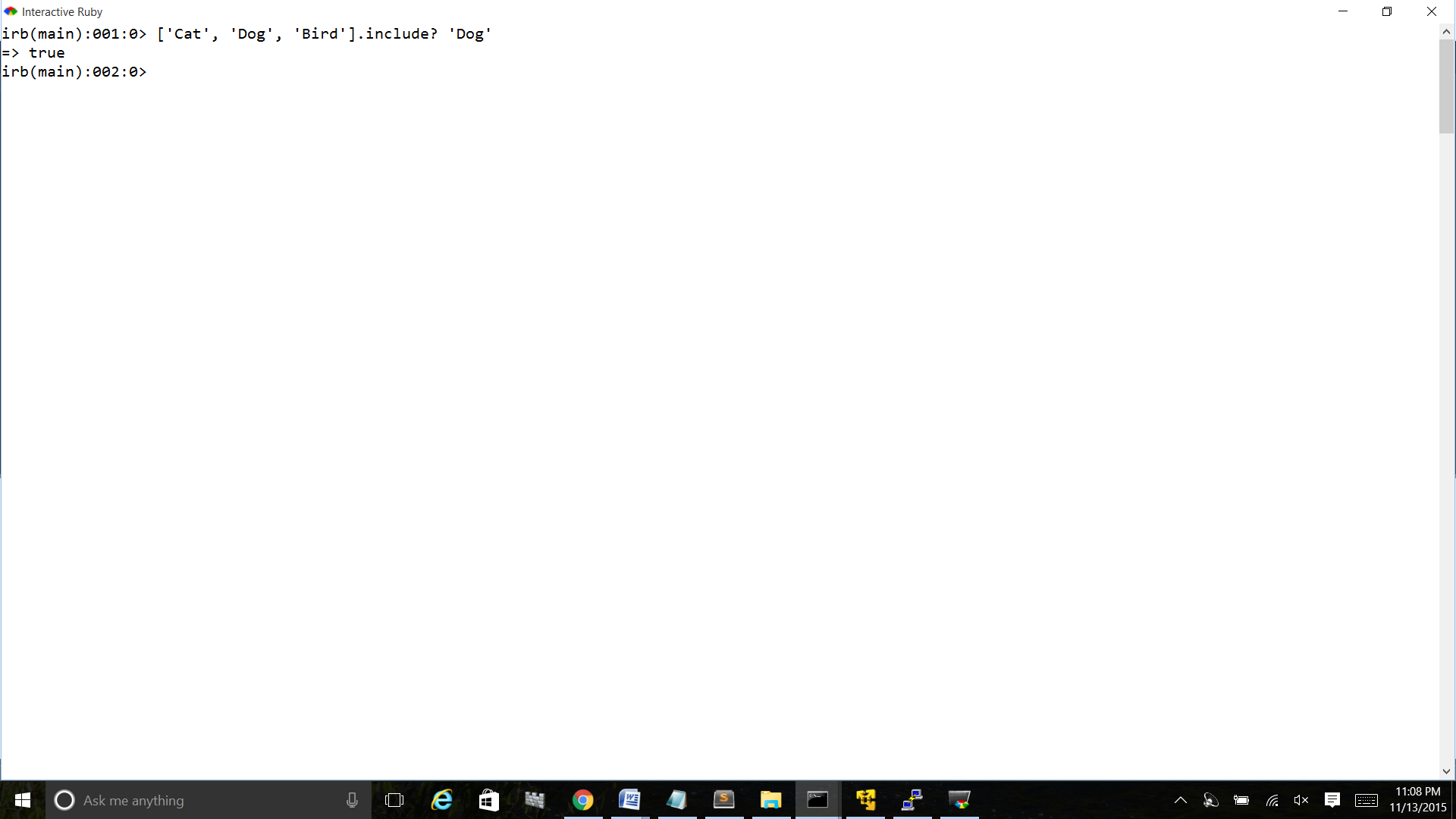
end



6. Write a Ruby program that loops through a array and checks if a pattern exists in the array elements

Ans: #!/usr/bin/env ruby

['Cat', 'Dog', 'Bird'].include? 'Dog'



7. Write your own Ruby program using a Hash that loops through :

Print all Values while looping with Keys

Print all Keys while looping through Values

Print Keys, Values as pair.

Ans:

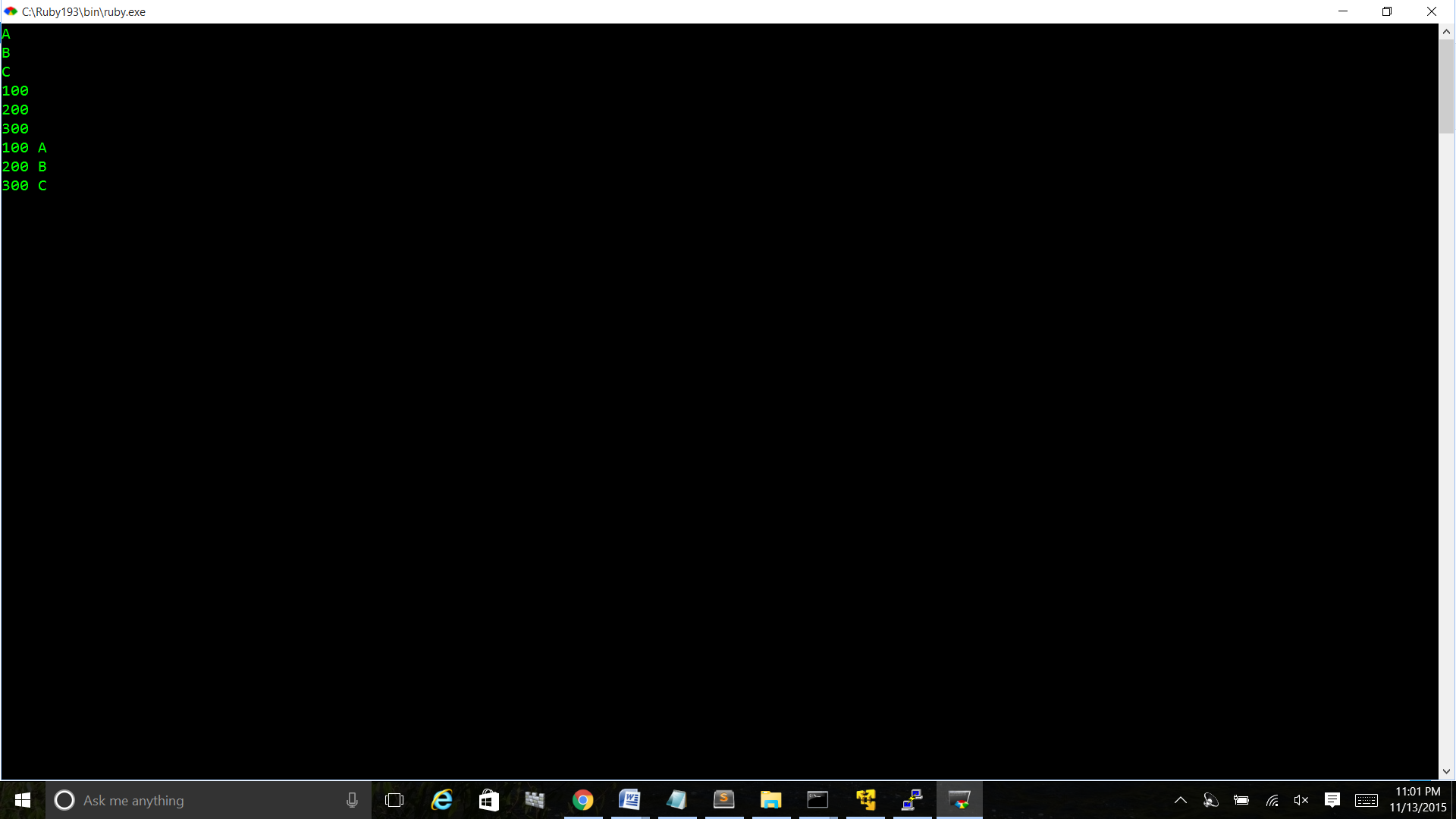
numbers = {100 => "A", 200 => "B", 300 => "C"}

puts numbers.values

puts numbers.keys

numbers.each {|k, v| print k, " ", v, "\n"}

gets



8. Write a Ruby program that takes number as input and recursively calculates the power of 2 until the calculated number is less than 10000 and prints the maximum power for that number.

9. Ruby program to convert Celsius temperature to Fahrenheit

Ans: #!/usr/bin/env ruby

puts "Current temperature:"

current\_temperature = gets.strip.to\_f

puts "Is this in celsius or Fahrenheit? [C/F]"

original\_temperature = gets.strip[0].downcase

if original\_temperature == 'c'

new\_temperature = (current\_temperature \* 5 / 9) + 32

puts "New temperature: #{new\_temperature} F"

elsif original\_temperature == 'f'

new\_temperature = (current\_temperature - 32) \* 9 / 5

puts "New temperature: #{new\_temperature} C"

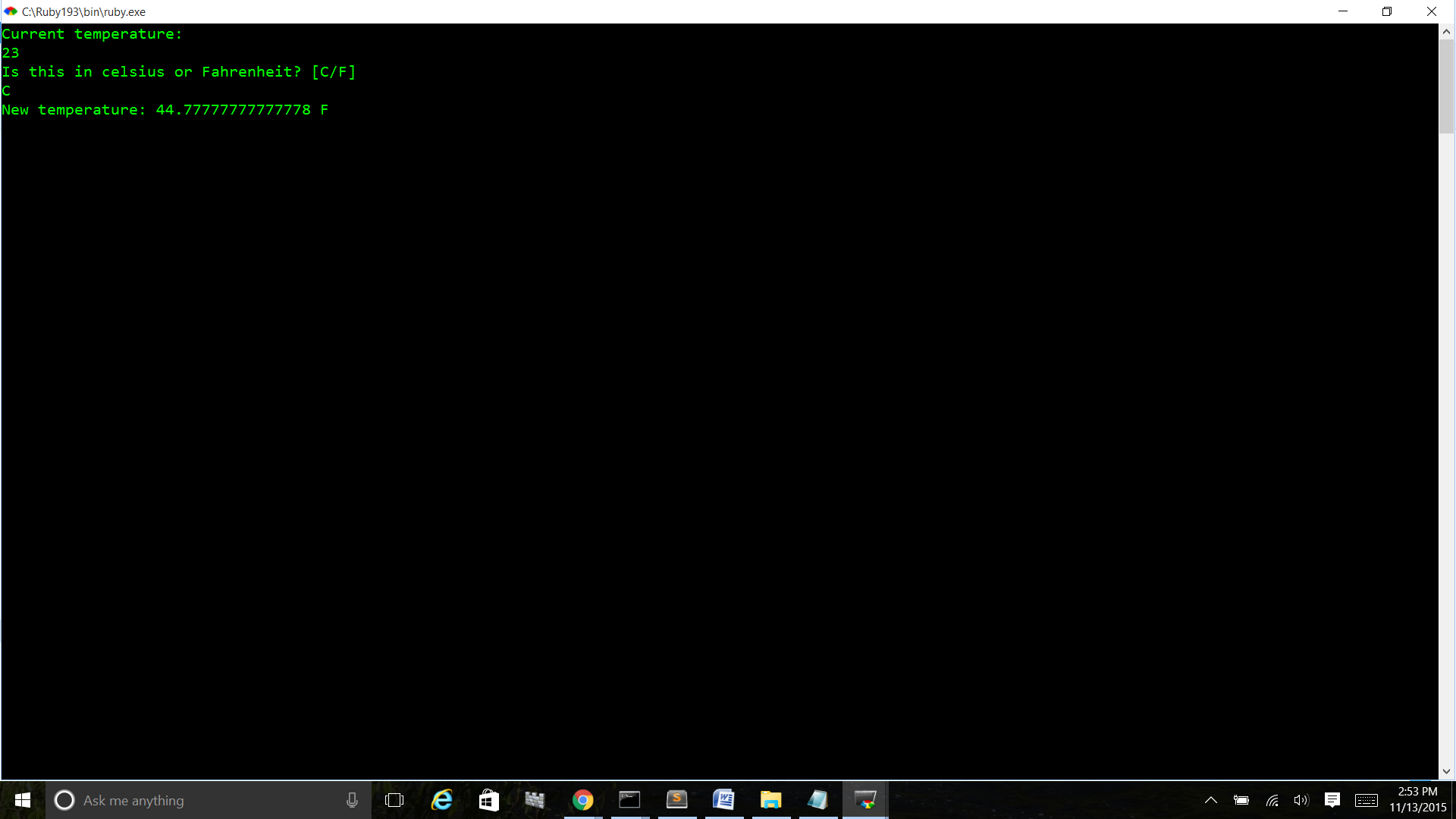
else

puts "Don't recognise temperature scale: #{original\_temperature}"

end

gets

**OutPut:**



10. Write a program to create a Calculator class with add(), substract(), multiply() and divide(), then take two numbers and choice of operation from user and display output using objects

Ans:

class Calculator

attr\_accessor :in1,:in2

def initialize(in1,in2)

@in1=in1

@in2=in2

end

def add()

puts @in1 + @in2

end

def sub()

puts @in1 - @in2

end

def mul()

puts @in1 \* @in2

end

def div()

puts @in1 % @in2

end

end

puts "enter first number"

@in1=gets.to\_i

puts "enter second number"

@in2=gets.to\_i

c=Calculator.new(@in1,@in2)

puts "Enter the Operator"

op=gets.chomp

case op

when "+"

c.add

when "-"

c.sub

when "\*"

c.mul

when "/'"

c.div

else

puts "invalid operator"

end

gets

