```ruby
# Blocks and Closures

# Proc

def get_a_multiplier(factor)
  proc { |n| n * factor }
end

times3 = get_a_multiplier(3)
times5 = get_a_multiplier(5)

times3.call(12).send :display                  #=> 36
puts
times5.call(5).send :display                   #=> 25
puts
times3.call(times5.call(4)).send :display      #=> 60
puts

# yield
def my_if(cond)
  yield if cond
end

a = 5
my_if(a < 10) do
  puts 'a is less than 10'
  a += 1
end
p a

# lambda vs proc
my_lambda = ->(x, y) { puts x + y }
my_proc = proc { |x, y| puts x + y }

# works as expected, prints 6
my_proc.call(1, 5, 11)

# an ArgumentError exception is thrown because the extra argument gets caught by
# arity of lambdas
# my_lambda.call(1, 5, 11)

def return_from_a_proc
  ret = proc { return 'Here I go from a proc' }
  ret.call
  'This is not reached'
end

# prints "Here I go from a proc"
puts return_from_a_proc

def return_from_a_lambda
  ret = -> { return 'Here I go from a lambda' }
  ret.call
  'This is printed'
end

# prints "This is printed"
puts return_from_a_lambda

# blocks
myarray = [1, 2, 3, 4, 5]

myarray.each do |item|
  item.send :display
end
puts
def method
  val = yield 0
  puts "value of yield0 : #{val}"
  val = yield 1
  puts "value of yield1 : #{val}"
end

method { |i| "return_val = #{i}" }

myarray = [1, 2, 3, 4, 5]
myarray.send :display
puts
myarray.find_all { |array_item| array_item > 3 }.send :display    # [4, 5]
puts
myarray.map { |array_item| array_item * 2 }.send :display         # [2, 4, 6, 8, 10]
puts
```