



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.03 «Прикладная информатика»

О Т Ч Е Т

по лабораторной работе № 6

Название: Ruby. Массивы и строковая обработка.

Дисциплина: Языки Интернет-программирования

Студент

ИУ6-34Б

(Группа)

20.10.2022

(Подпись, дата)

С. А. Рахманов

(И.О. Фамилия)

Преподаватель

Д. В. Малахов

(Подпись, дата)

(И.О. Фамилия)

Москва, 2022

Цель работы: получение навыков программирования на языке Ruby с использованием функционального стиля программирования, использование средств проверки соответствия стиля программирования.

Задание:

Все консольные приложения Ruby следует реализовывать в виде трех отдельных файлов:

1. основная программа;
2. программа для взаимодействия с пользователем через консоль;
3. программа для автоматического тестирования на основе `MiniTest::Unit` или `RSpec`. Везде, где это возможно, данные для проверки должны формироваться автоматически по правилам, указанным в задании.

При реализации программ везде, где это возможно, следует избегать использования циклов `for`, `do`, `while`. Вместо них используйте методы из примеси `Enumerable`.

Все тексты программ должны быть проверены на соответствие стилю программирования Ruby при помощи `rubocop.ru` или `reek`.

ЛР 6

Часть 1

Решить задачу, организовав цикл. Вычислить определенный интеграл методом трапеций: $\int_{\pi/4}^{\pi/3} tg^2 x \, dx$. Отрезок интегрирования разбить на $n =$

20, 30, 40 частей. Точное значение: $\sqrt{3} - \frac{\pi}{12} - 1$. Оценить погрешность вычислений при различных n .

Часть 2

Решить предыдущее задание с помощью Enumerable или Enumerator.

Часть 3

Составить метод `intg` вычисления определенного интеграла по формуле прямоугольников:

$S = \frac{b-a}{n} \sum_{i=1}^n f(x_i)$, где n – количество отрезков разбиения. В основной

программе использовать метод `intg` для вычисления интегралов: $\int_{0,1}^1 \frac{\sin x}{x} \, dx$

и $\int_1^2 \frac{tg(x+1)}{x+1} \, dx$.

Реализовать вызов метода двумя способами: в виде передаваемого `lambda`-выражения и в виде блока.

Тексты программ

Часть 1

interface.rb

```
# frozen_string_literal: true
```

```
require './main'
```

```
loop do
```

```
  print 'Start? [y/n]'
```

```
  start = gets.chomp
```

```
  if start == 'y'
```

```
    n = 20
```

```
    print "integral (n = #{n}) = #{calc(n).round(6)}    accuracy = #{(calc(n) - KEY).round(6)}\n"
```

```
    n = 30
```

```
    print "integral (n = #{n}) = #{calc(n).round(6)}    accuracy = #{(calc(n) - KEY).round(6)}\n"
```

```
    n = 40
```

```

    print "integral (n = #{n}) = #{calc(n).round(6)}    accuracy = #{(calc(n)
- KEY).round(6)}\n"
  else
    exit
  end
end
end

```

main.rb

```

# frozen_string_literal: true

KEY = 3**0.5 - Math::PI / 12 - 1

def calc(num)
  y = ->(x) { Math.tan(x)**2 }
  a = x_cur = x_next = Math::PI / 4
  b = Math::PI / 3
  step = (b - a) / num
  res = 0
  (0...num).each do |i|
    x_next += step
    y_cur = y.call x_cur
    y_next = y.call x_next
    res += (y_cur + y_next) / 2 * step
    x_cur += step
  end
  res
end

```

test.rb

```

# frozen_string_literal: true

require 'minitest/autorun'
require './main'

# Test Class
class TestTree < Minitest::Test
  # first test
  def test_1
    data = Array.new(10) { rand(10..100) }
    10.times do |i|
      assert_in_delta(KEY, calc(data[i]), 0.001,
        ["Incorrect value function calc(#{data[i]}) ((instead
of #{calc(data[i]}))")])
    end
  end
end
end

```

Часть 2

interface.rb

```

# frozen_string_literal: true

require './main'

loop do
  print 'Start? [y/n]'
  start = gets.chomp
  if start == 'y'
    n = 20
    print "integral (n = #{n}) = #{calc(n).round(6)}    accuracy =
#{(calc(n).to_f - KEY).round(6)}\n"
    n = 30
    print "integral (n = #{n}) = #{calc(n).round(6)}    accuracy =
#{(calc(n).to_f - KEY).round(6)}\n"
  end
end

```

```

        n = 40
        print "integral (n = #{n}) = #{calc(n).round(6)}    accuracy =
#{(calc(n).to_f - KEY).round(6)}\n"
      else
        exit
      end
    end
  end
end

```

main.rb

```

# frozen_string_literal: true

KEY = 3**0.5 - Math::PI / 12 - 1
A = Math::PI / 4
B = Math::PI / 3
Y = ->(x) { Math.tan(x)**2 }

# with Enumerable
def calc(num)
  a = A
  b = B
  my_step = (b - a) / num
  a += my_step / 2
  (a...b).step(my_step).each.inject(0) { |acc, x| acc + Y.call(x) *
my_step }
end

# with Enumerator
def calc1(num)
  values = Enumerator.new do |val|
    a = A
    b = B
    step = (b - a) / num
    a += step / 2
    num.times do
      val << Y.call(a) * step
      a += step
    end
  end
  values.sum
end

```

test.rb

```

# frozen_string_literal: true

require 'minitest/autorun'
require './main.rb'

# Test Class
class TestTree < Minitest::Test
  # first test
  def test_1
    data = Array.new(10) { rand(10..100) }
    data.each do |d|
      # res = calc(d)
      assert_in_delta(KEY, calc(d), 0.001, ["Incorrect value function
calc(#{d}) (instead of #{KEY})"])
    end
  end
end

```

Часть 3

interface.rb

```

# frozen_string_literal: true

```

```

require './main.rb'

loop do
  print "\nStart? [y/n]"
  start = gets.chomp
  if start == 'y'

    puts "Choose function: \n1. sin(x)/x \n" + "2. tan(x+1)/(x+1) \n"
    func_num = gets.chomp
    puts "Choose creation method: \n1. lambda\n" + "2. yield\n"
    method_num = gets.chomp

    case func_num
    when '1' then func = 'sin(x)/x'
      case method_num
      when '1' then block = ->(x) { Math.sin(x) / x }
      when '2' then block = proc { |x| Math.sin(x) / x }
      # some_func = func_sin
      else abort 'Error'
      end
    when '2' then func = 'tan(x+1)/(x+1)'
      case method_num
      when '1' then block = ->(x) { Math.tan(x + 1) / (x + 1) }
      when '2' then block = proc { |x| Math.tan(x + 1) / (x + 1) }
      # some_func = func_tan
      else abort 'Error'
      end
    else abort 'Error'
    end

    puts '*lil - low integration limit, uil - upper integration limit, n -
number of steps' + "\n"
    puts 'Input lil'
    lil = gets.chomp.to_f
    puts 'Input uil'
    uil = gets.chomp.to_f
    puts 'Input n'
    n = gets.chomp.to_i

    puts 'Standart form of output: ' + "\u222b".encode('utf-8') + '(lil,
uil, func)'
    print "\u222b".encode('utf-8') + "({lil}, {uil}, {func})" + " =
#{intg(lil, uil, n, &block)}" + "\n"

    else exit
    end
  end
end

```

main.rb

```

# frozen_string_literal: true

def intg(lil, uil, num, &block)
  step = (uil - lil) / num
  summa_func = 0
  x = lil + step / 2 # (step / 2 give more accuracy)
  (0...num).each do # (more accuracy then ..)
    summa_func += block.call x
    x += step
  end
  (uil - lil) / num * summa_func
end

```

test.rb

```
# frozen_string_literal: true

require 'minitest/autorun'
require './main.rb'

# Test Class
class TestTree < Minitest::Test
  # first test

  def test_1
    data = [
      [0.1, 1, 100, ->(x) { Math.sin(x) / x }, 0.846139],
      [0.1, 1, 100, proc { |x| Math.sin(x) / x }, 0.846139],
      [1.0, 2.0, 100, ->(x) { Math.tan(x + 1) / (x + 1) }, -0.376871],
      [1.0, 2.0, 100, proc { |x| Math.tan(x + 1) / (x + 1) }, -0.376871],

      [0.1, 1, 500, ->(x) { Math.sin(x) / x }, 0.846139],
      [0.1, 1, 500, proc { |x| Math.sin(x) / x }, 0.846139],
      [1.0, 2.0, 500, ->(x) { Math.tan(x + 1) / (x + 1) }, -0.376871],
      [1.0, 2.0, 500, proc { |x| Math.tan(x + 1) / (x + 1) }, -0.376871],

      [0.1, 1, 1000, ->(x) { Math.sin(x) / x }, 0.846139],
      [0.1, 1, 1000, proc { |x| Math.sin(x) / x }, 0.846139],
      [1.0, 2.0, 1000, ->(x) { Math.tan(x + 1) / (x + 1) }, -0.376871],
      [1.0, 2.0, 1000, proc { |x| Math.tan(x + 1) / (x + 1) }, -0.376871],

      # some another tests
      [0, Math::PI / 2, 100, ->(x) { Math.sin(x) }, 1],
      [0, Math::PI / 2, 100, proc { |x| Math.sin(x) }, 1],
      [0, Math::PI / 2, 100, ->(x) { Math.sin(x) * Math.cos(x) }, 0.5],
      [0, Math::PI / 2, 100, proc { |x| Math.sin(x) * Math.cos(x) }, 0.5],
      [0, Math::PI / 2, 500, ->(x) { Math.sin(x) * Math.cos(x) }, 0.5],
      [0, Math::PI / 2, 500, proc { |x| Math.sin(x) * Math.cos(x) }, 0.5],
      [0, 2.0, 1000, ->(x) { x**10 + 5 * x**4 }, 218.18],
      [0, 2.0, 1000, proc { |x| x**10 + 5 * x**4 }, 218.18],
      [0, 2.0, 400, ->(x) { x**10 + 5 * x**4 }, 218.18],
      [0, 2.0, 535, proc { |x| x**10 + 5 * x**4 }, 218.18]
    ]
    data.each do |d|
      temp_str = intg(d[0], d[1], d[2], &d[3])
      print d
      assert_in_delta(d[4], temp_str, 0.01, ["Incorrect work of function
intg() = #{intg(d[0], d[1], d[2], &d[3])} (instead of #{d[4]})"])
    end
  end
end
```

Результаты выполнения

Часть 1

```
Терминал
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part1$ ruby interface.rb
Start? [y/n]y
integral (n = 20) = 0.470392    accuracy = 0.000141
integral (n = 30) = 0.470314    accuracy = 6.3e-05
integral (n = 40) = 0.470287    accuracy = 3.5e-05
Start? [y/n]n
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part1$ ruby test.rb
Run options: --seed 13663

# Running:

.

Finished in 0.003449s, 289.9453 runs/s, 2899.4533 assertions/s.
1 runs, 10 assertions, 0 failures, 0 errors, 0 skips
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part1$
```

Часть 2

```
Терминал
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6$ cd part2
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part2$ ruby interface.rb
Start? [y/n]y
integral (n = 20) = 0.470181    accuracy = -7.0e-05
integral (n = 30) = 0.47022    accuracy = -3.1e-05
integral (n = 40) = 0.470234    accuracy = -1.8e-05
Start? [y/n]n
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part2$ ruby test.rb
Run options: --seed 53396

# Running:

.

Finished in 0.003118s, 320.7573 runs/s, 3207.5730 assertions/s.
1 runs, 10 assertions, 0 failures, 0 errors, 0 skips
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part2$
```

Часть 3


```
Терминал
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part3$ ruby interface.rb

Start? [y/n]y
Choose function:
1. sin(x)/x
2. tan(x+1)/(x+1)
1
Choose creation method:
1. lambda
2. yield
1
*lil - low integration limit, uil - upper integration limit, n - number of steps
Input lil
0.1
Input uil
1
Input n
100
Standart form of output: f(lil, uil, func)
f(0.1, 1.0, sin(x)/x) = 0.8461395133169102

Start? [y/n]y
Choose function:
1. sin(x)/x
2. tan(x+1)/(x+1)
2
Choose creation method:
1. lambda
2. yield
2
*lil - low integration limit, uil - upper integration limit, n - number of steps
Input lil
1
Input uil
2
Input n
500
Standart form of output: f(lil, uil, func)
f(1.0, 2.0, tan(x+1)/(x+1)) = -0.3768701444956157

Start? [y/n]n
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part3$ ruby test.rb
Run options: --seed 14888

# Running:

.

Finished in 0.006081s, 164.4535 runs/s, 3617.9759 assertions/s.
1 runs, 22 assertions, 0 failures, 0 errors, 0 skips
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part3$
```

Результаты проверки анализатором rubocop

Часть 1

```
Терминал
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part1$ rubocop main.rb
Inspecting 1 file
.

1 file inspected, no offenses detected
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part1$ rubocop test.rb
Inspecting 1 file
.

1 file inspected, no offenses detected
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part1$ rubocop interface.rb
Inspecting 1 file
.

1 file inspected, no offenses detected
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part1$
```

Часть 2

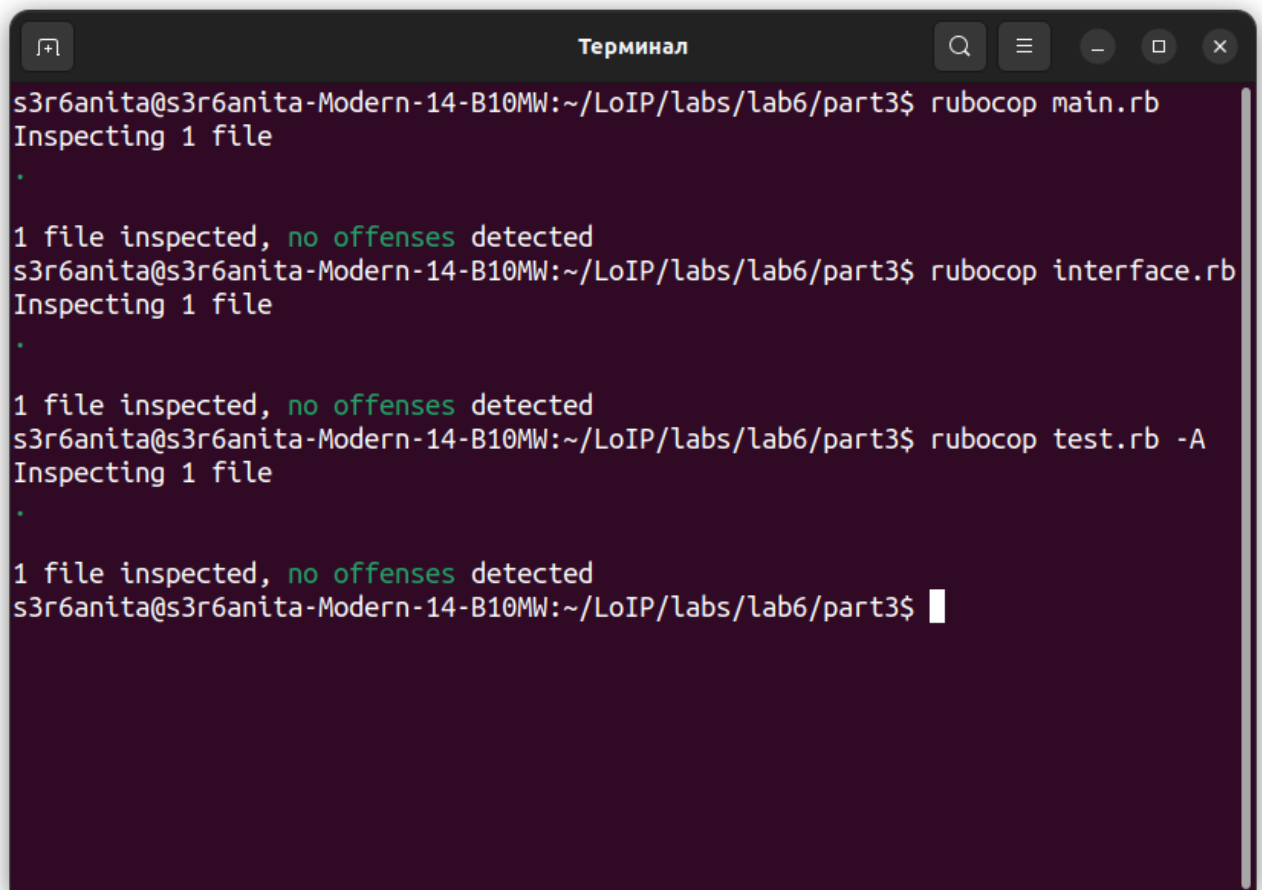
```
Терминал
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part2$ rubocop main.rb
Inspecting 1 file
.

1 file inspected, no offenses detected
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part2$ rubocop test.rb
Inspecting 1 file
.

1 file inspected, no offenses detected
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part2$ rubocop interface.rb
Inspecting 1 file
.

1 file inspected, no offenses detected
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part2$
```

Часть 3



```
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part3$ rubocop main.rb
Inspecting 1 file
.

1 file inspected, no offenses detected
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part3$ rubocop interface.rb
Inspecting 1 file
.

1 file inspected, no offenses detected
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part3$ rubocop test.rb -A
Inspecting 1 file
.

1 file inspected, no offenses detected
s3r6anita@s3r6anita-Modern-14-B10MW:~/LoIP/labs/lab6/part3$
```

Вывод: получил навыки программирования на языке Ruby с использованием функционального стиля программирования, получил навыки использования rubocop и проверил написанный код на соответствие стилю программирования на Ruby.