|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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»**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 6 |

**Название:**

Ruby. Массивы и строковая обработка.

**Дисциплина:** Языки Интернет-программирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-34Б |  | 20.10.2022 | С. А. Рахманов |
|  | (Группа) |  | (Подпись, дата) | (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.

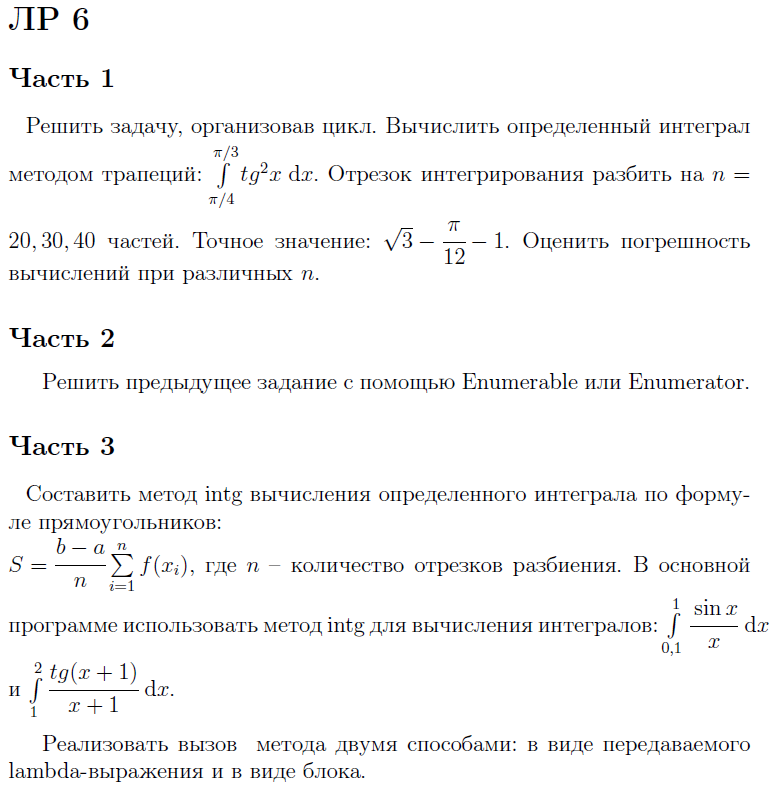
При реализации программ везде, где это возможно, следует избегать

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.



**Тексты программ**

**Часть 1**

*interface.rb*

# frozen\_string\_literal: true

require './main'

loop do

print 'Start? [y/n]'

start = gets.chop

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

*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

**Часть 2**

*interface.rb*

# frozen\_string\_literal: true

require './main'

loop do

print 'Start? [y/n]'

start = gets.chop

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"

n = 40

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

else

exit

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.chop

if start == 'y'

puts "Choose function: \n1. sin(x)/x \n" + "2. tan(x+1)/(x+1) \n"

func\_num = gets.chop

puts "Choose creation method: \n1. lambda\n" + "2. yield\n"

method\_num = gets.chop

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.chop.to\_f

puts 'Input uil'

uil = gets.chop.to\_f

puts 'Input n'

n = gets.chop.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

*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]})"])

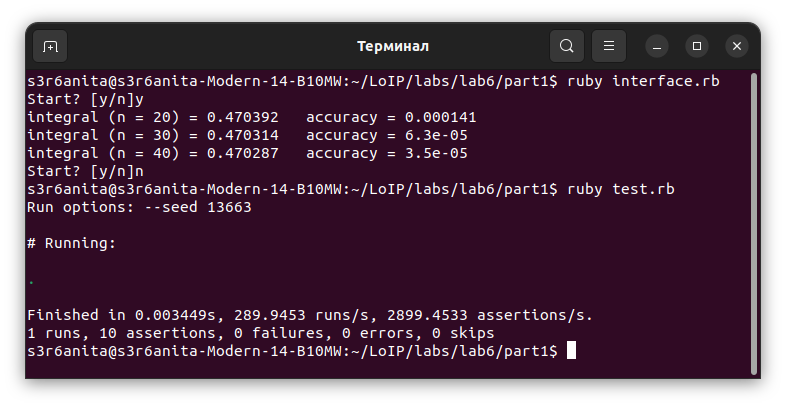
end

end

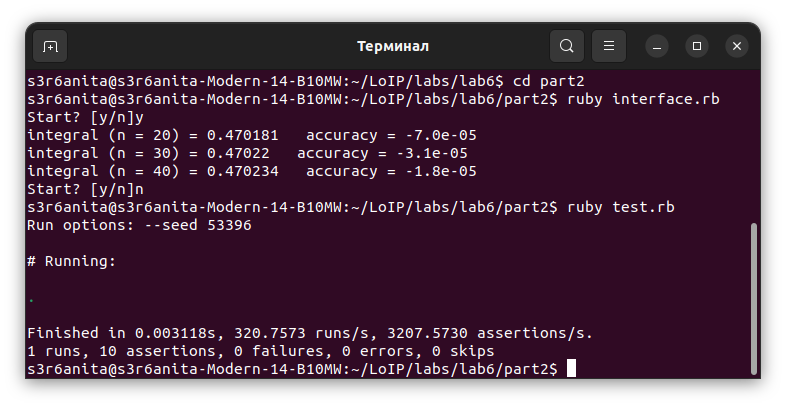
end

**Результаты выполнения**

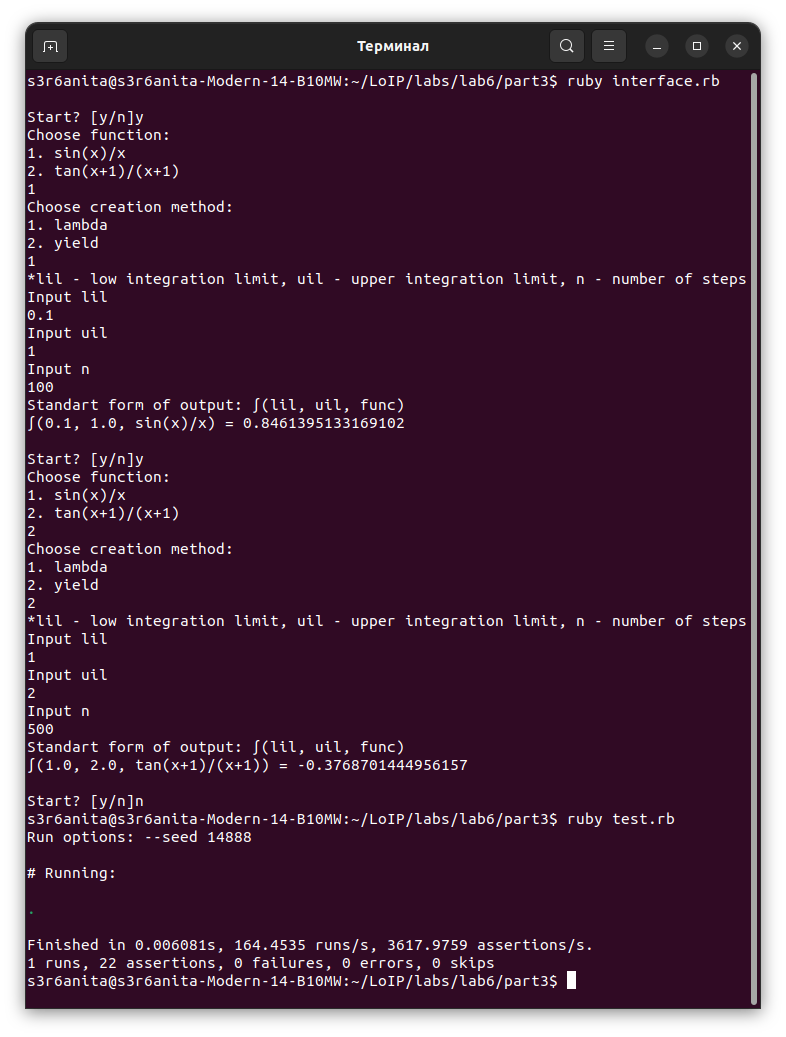
**Часть 1**

****

**Часть 2**

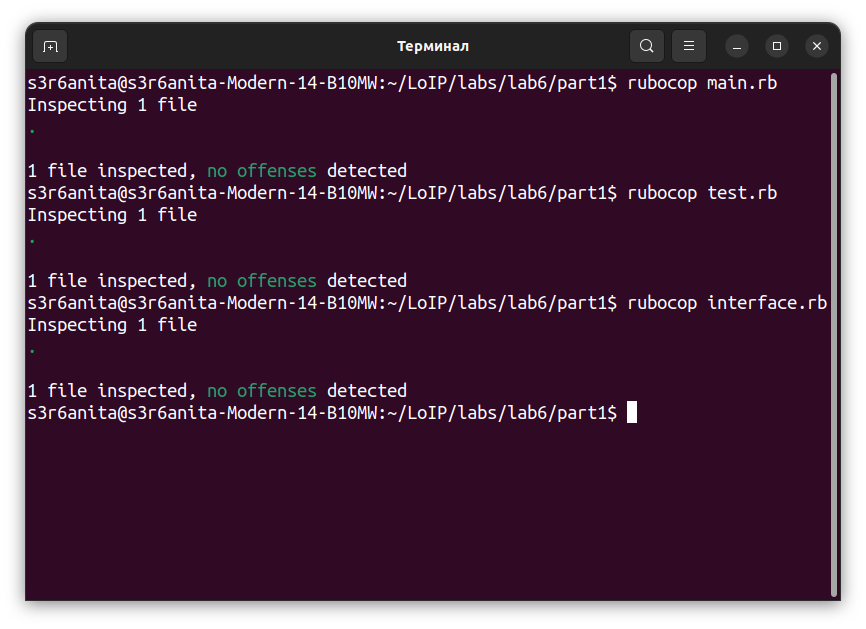
****

**Часть 3**

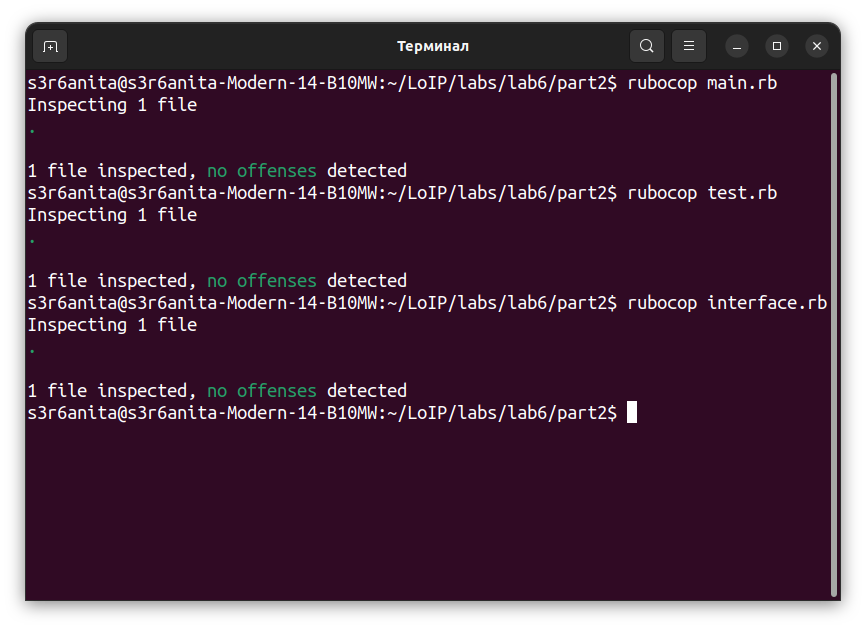
****

**Результаты проверки анализатором rubocop**

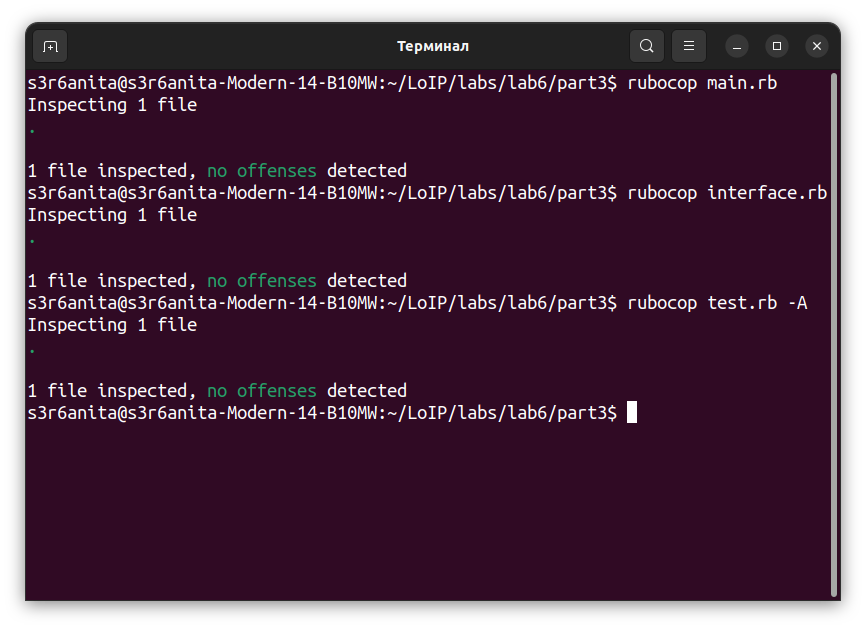
**Часть 1**

****

**Часть 2**

****

**Часть 3**

****

**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.