|  |  |
| --- | --- |
| 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 **09.03.01 Информатика и вычислительная техника**

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 6 |

**Название:**

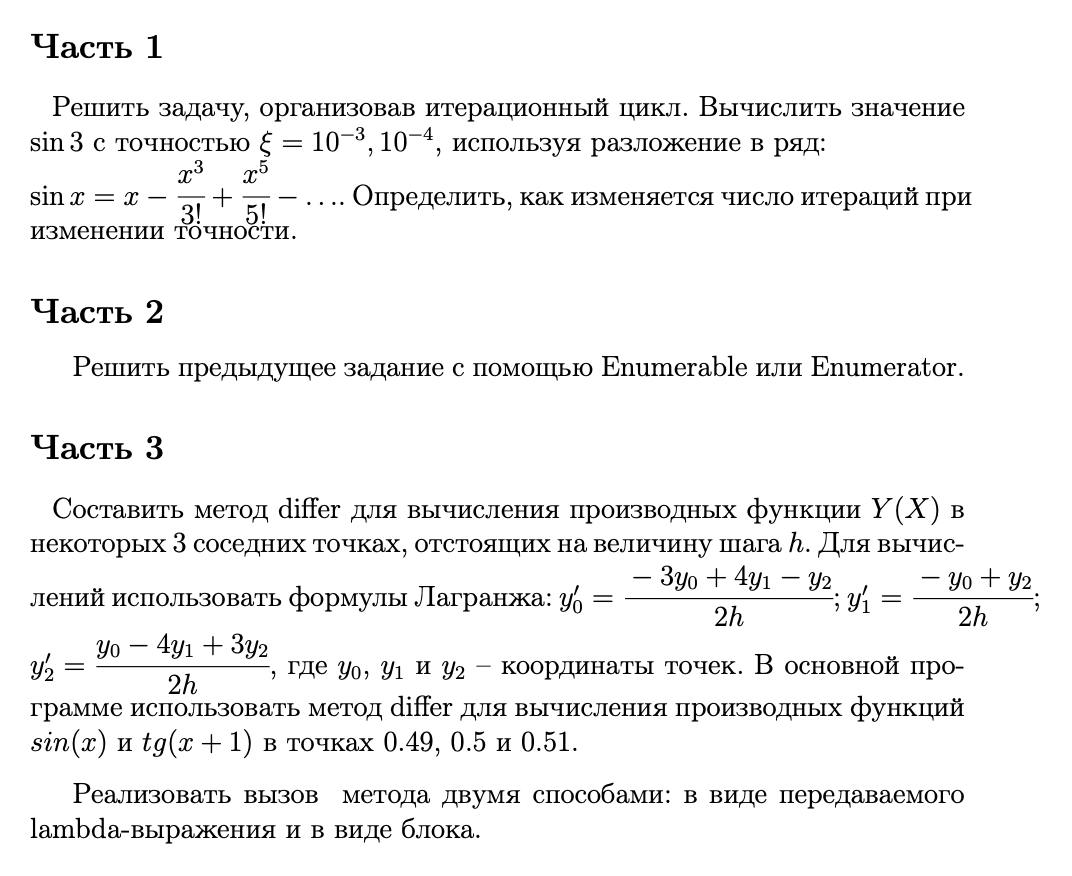
[Классы, модули, примеси.Файл](http://e-learning.bmstu.ru/moodle/mod/resource/view.php?id=993)

**Дисциплина:** Языки интернет-программирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-32Б |  |  | И.А. Люляев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  |  |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

**Условие**



**Тексты программ**

**Calc\_sin.rb**

# frozen\_string\_literal: true

# class with calculation

class Calc

SIN\_X = 3

EPS\_3 = 0.001

EPS\_4 = 0.0001

PRECISE\_VALUE = Math.sin(3)

def calc\_sin(eps)

calc\_value = SIN\_X

is\_sum = false

n\_dependant = 3

count = 0

while (PRECISE\_VALUE - calc\_value).abs > eps

calculation = (SIN\_X\*\*n\_dependant) / factorial(n\_dependant).to\_f

if is\_sum

calc\_value += calculation

else

calc\_value -= calculation

end

is\_sum = !is\_sum

n\_dependant += 2

count += 1

end

p 'Result calculated in ' + count.to\_s + ' iterations'

p 'Calculated value: ' + calc\_value.to\_s + ' (precise value is 0.1411200080598672)'

calc\_value

end

def factorial(nnn)

if nnn.zero?

1

else

nnn \* factorial(nnn - 1)

end

end

end

**calc\_sin\_enum.rb**

# frozen\_string\_literal: true

# class with calculation

class CalcEnum

SIN\_X = 3

EPS\_3 = 0.001

EPS\_4 = 0.0001

PRECISE\_VALUE = Math.sin(3)

def print(eps)

p 'Calculation:'

p calc\_arr = CalcEnum.new.calc\_sin\_enum.take\_while do |calc\_value|

(CalcEnum::PRECISE\_VALUE - calc\_value).abs >= eps \* 0.1

end

puts 'Precise value of sin(3): ' + PRECISE\_VALUE.to\_s

calc\_arr[calc\_arr.size - 1]

end

def calc\_sin\_enum

Enumerator.new do |yielded|

calc\_value = SIN\_X

is\_sum = false

n\_dependant = 3

count = 0

loop do

yielded.yield calc\_value

calculation = (SIN\_X\*\*n\_dependant) / factorial(n\_dependant).to\_f

if is\_sum

calc\_value += calculation

else

calc\_value -= calculation

end

is\_sum = !is\_sum

n\_dependant += 2

count += 1

end

p calc\_value

end

end

def factorial(nnn)

if nnn.zero?

1

else

nnn \* factorial(nnn - 1)

end

end

end

**differ.rb**

# frozen\_string\_literal: true

POINT\_0 = 0.49

POINT\_1 = 0.50

POINT\_2 = 0.51

STEP = (POINT\_1 - POINT\_0).abs

# class with calculation

class Differ

def differ\_with\_block

dif\_point0 = (-3 \* yield(POINT\_0) + 4 \* yield(POINT\_1) - yield(POINT\_2)) / (2 \* STEP).to\_f

dif\_point1 = (-yield(POINT\_0) + yield(POINT\_2)) / (2 \* STEP).to\_f

dif\_point2 = (yield(POINT\_0) - 4 \* yield(POINT\_1) + 3 \* yield(POINT\_2)) / (2 \* STEP).to\_f

p [dif\_point0.round(2), dif\_point1.round(2), dif\_point2.round(2)]

end

def differ\_with\_lambda(math\_func)

dif\_point0 = (-3 \* math\_func.call(POINT\_0) + 4 \* math\_func.call(POINT\_1) - math\_func.call(POINT\_2)) /

(2 \* STEP).to\_f

dif\_point1 = (-math\_func.call(POINT\_0) + math\_func.call(POINT\_2)) / (2 \* STEP).to\_f

dif\_point2 = (math\_func.call(POINT\_0) - 4 \* math\_func.call(POINT\_1) + 3 \* math\_func.call(POINT\_2)) /

(2 \* STEP).to\_f

p [dif\_point0.round(2), dif\_point1.round(2), dif\_point2.round(2)]

end

end

**client.rb**

# frozen\_string\_literal: true

require './calc\_sin.rb'

require './calc\_sin\_enum.rb'

require './differ.rb'

puts 'Enter function you want to execute: calc\_sin, calc\_sin\_enum or differ?'

arg = gets.chomp

case arg

when 'calc\_sin'

puts 'choose precision: 10^-3 or 10^-4'

case gets.chomp

when '-3'

Calc.new.calc\_sin(Calc::EPS\_3)

when '-4'

Calc.new.calc\_sin(Calc::EPS\_4)

else

p 'invalid input'

end

when 'calc\_sin\_enum'

puts 'choose precision: 10^-3 or 10^-4'

case gets.chomp

when '-3'

CalcEnum.new.print(CalcEnum::EPS\_3)

when '-4'

CalcEnum.new.print(CalcEnum::EPS\_4)

else

p 'invalid input'

end

when 'differ'

puts 'block or lambda'

case gets.chomp

when 'block'

puts 'sin or tg'

case gets.chomp

when 'sin'

Differ.new.differ\_with\_block do |x|

Math.sin(x)

end

when 'tg'

Differ.new.differ\_with\_block do |x|

Math.tan(x)

end

else

p 'invalid input'

end

when 'lambda'

puts 'sin or tg'

case gets.chomp

when 'sin'

Differ.new.differ\_with\_lambda(->(x) { Math.sin(x) })

when 'tg'

Differ.new.differ\_with\_lambda(->(x) { Math.tan(x) })

else

p 'invalid input'

end

else

p 'invalid input'

end

else

p 'invalid input'

end

**test.rb**

# frozen\_string\_literal: true

require './calc\_sin.rb'

require './calc\_sin\_enum.rb'

require './differ.rb'

require 'minitest/autorun'

# Class for setting any tests on functions

class Tests < MiniTest::Test

def setup; end

def test\_calc\_sin

assert\_in\_delta(Math.sin(3), Calc.new.calc\_sin(Calc::EPS\_3), Calc::EPS\_3)

assert\_in\_delta(Math.sin(3), Calc.new.calc\_sin(Calc::EPS\_4), Calc::EPS\_4)

end

def test\_calc\_sin\_enum

assert\_in\_delta(Math.sin(3), CalcEnum.new.print(CalcEnum::EPS\_3), Calc::EPS\_3)

assert\_in\_delta(Math.sin(3), CalcEnum.new.print(CalcEnum::EPS\_4), Calc::EPS\_4)

end

# :reek:UncommunicativeVariableName

def test\_differ\_lambda

assert\_equal([0.88, 0.88, 0.87], Differ.new.differ\_with\_lambda(->(x) { Math.sin(x) }))

assert\_equal([1.28, 1.30, 1.31], Differ.new.differ\_with\_lambda(->(x) { Math.tan(x) }))

end

# :reek:UncommunicativeVariableName

def test\_differ\_block

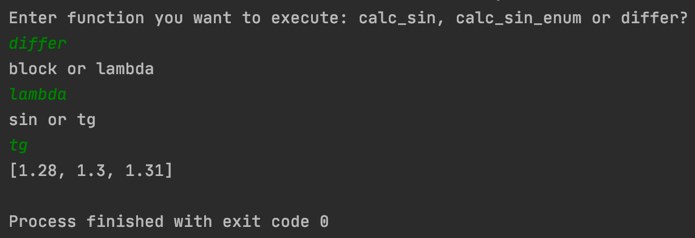
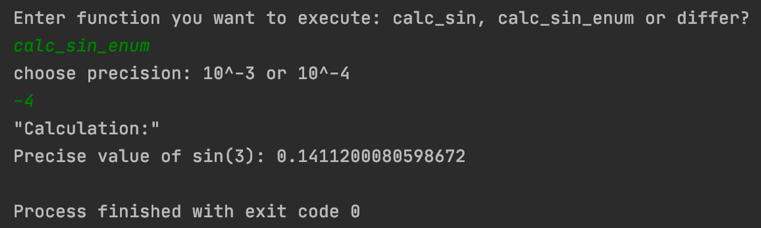
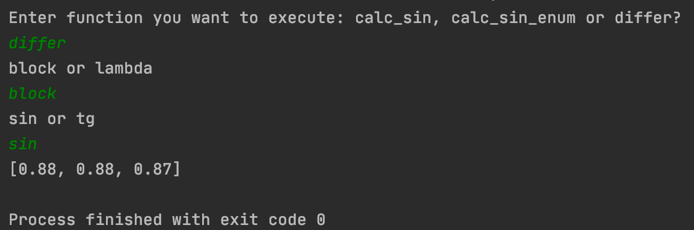
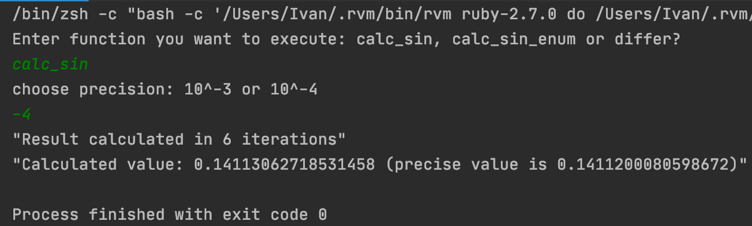
assert\_equal([0.88, 0.88, 0.87], Differ.new.differ\_with\_block { |x| Math.sin(x) })

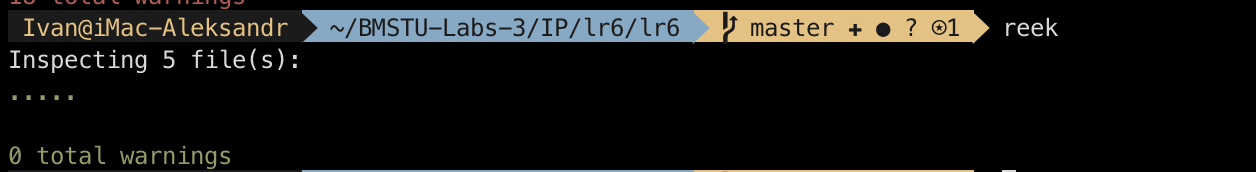
assert\_equal([1.28, 1.30, 1.31], Differ.new.differ\_with\_block { |x| Math.tan(x) })

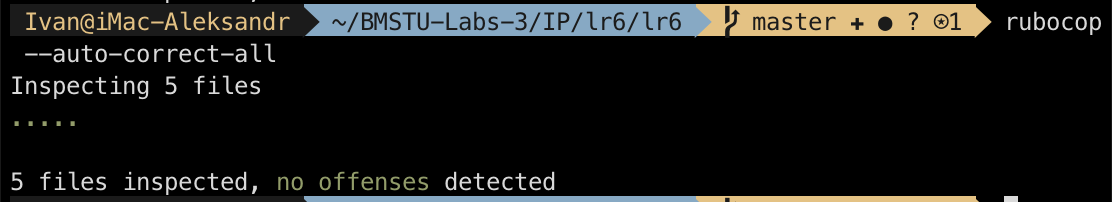
end

def teardown; end

end

**Результаты выполнения**

**Результаты проверки анализаторами rubocop и reek**

****

**Вывод:** я научился работать с Enumerator, а также block и lambda