Homework#5 Ruby – Documentation

CSc 600-01

Dr. Jozo Dujmovic

5.11.18

Keawa Rozet

916796184

Code available at: <https://github.com/krozet/CSC-600-Ruby>

Question #1 Prompt

1. Write a single Ruby demo program that illustrates the use of all main Ruby iterators (loop, while, until, for, upto, downto, times, each, map, step, collect, select, reject). The program should have a few lines illustrating loop, followed by a few lines illustrating while, and so on).

Code

# CSc 600: Homework 5

# Ruby

#

# Keawa Rozet

# Homework 5: Question #1

#while

counter = 2

while counter > 0

puts "while loop: #{counter}"

counter -= 1

end

puts

#until

counter = 4

until counter == 0

print "until loop: #{counter}\n"

counter /= 2

end

puts

#for

for i in 0..2

puts "for loop: #{i}"

end

puts

#upto

1.upto(3) {|x| puts "upto iterator: #{x}"}

puts

#downto

3.downto(1) do |y|

puts "downto iterator: #{y}"

end

puts

#times

numOfTimes = 2

numOfTimes.times {puts "times iterator: #{numOfTimes}"}

puts

#each

food = {1 => "apples", 2 => "bananas", 3 => "pears"}

food.each {|fruit| puts "each iterator: #{fruit}"}

puts

#map

sum = 0

(1..5).map do |x|

sum += x

end

puts "map iterator: #{sum}\n"

puts

#step

(0).step(1, 0.5) do |x|

puts "step iterator: #{x}"

end

puts

#collect

arr = [1,2,3,4].collect {|x| x\*2}

puts "collect iterator: #{arr}\n"

puts

#select

evenNums = (1..10).select {|x| x%2 == 0}

puts "select iterator: #{evenNums}"

puts

#reject

oddNums = (1..10).reject {|x| x%2 == 0}

puts "reject iterator: #{oddNums}"

Example

> loops.rb

while loop: 2

while loop: 1

until loop: 4

until loop: 2

until loop: 1

for loop: 0

for loop: 1

for loop: 2

upto iterator: 1

upto iterator: 2

upto iterator: 3

downto iterator: 3

downto iterator: 2

downto iterator: 1

times iterator: 2

times iterator: 2

each iterator: [1, "apples"]

each iterator: [2, "bananas"]

each iterator: [3, "pears"]

map iterator: 15

step iterator: 0.0

step iterator: 0.5

step iterator: 1.0

collect iterator: [2, 4, 6, 8]

select iterator: [2, 4, 6, 8, 10]

reject iterator: [1, 3, 5, 7, 9]

Question #2 Prompt

2. Write Ruby recognizer methods limited? and sorted? that expand the Ruby class Array. The expression array.limited?(amin,amax) should return true if amin ≤ a[i] ≤ amax for all values of i. The expression array.sorted? should return the following:  0 if the array is not sorted  +1 if a[0] ≤ a[1] ≤ a[2] ≤ … (increasing sequence)  -1 if a[0] ≥ a[1] ≥ a[2] ≥ … (decreasing sequence) Show examples of the use of this method.

About my Code

The limited? Function behaves in a straight forward manner by checking if each value lies within the interval.

The sorted? Function is slightly more complex by going through levels of analysis to determine what style of sorting has occurred. First, the array if checked for descending order, and then if it’s in ascending order. If neither, it says so.

Code

# CSc 600: Homework 5

# Ruby

#

# Keawa Rozet

# Homework 5: Question #2

class Array

def limited?(amin, amax)

self.each do |x|

unless amin <= x and x <= amax

return false

end

end

return true

end

def sorted?

if self.length > 1

if self[0] >= self[1]

#decreasing array

if self.length > 2

(2..self.length-1).each do |i|

#array is not sorted

unless self[i-1] >= self[i]

return 0

end

end

end

return +1

else

#increasing array

if self.length > 2

(2..self.length-1).each do |i|

#array is not sorted

unless self[i-1] <= self[i]

return 0

end

end

end

return -1

end

end

end

end

#test if incr or decr

def atest(value)

case value

when -1

puts "#{value} = decreasing"

when 0

puts "#{value} = not sorted"

when +1

puts "#{value} = increasing"

else

puts "Not an array"

end

end

#Examples

nums = [15,3,9,3,6,2,0,1]

puts "nums: #{nums}"

print "limited? to 1-15: "

puts nums.limited?(1, 15)

print "limited? to 0-15: "

puts nums.limited?(0, 15)

#not sorted

print "sorted?: #{nums}: "

atest(nums.sorted?)

#decreasing

nums = nums.sort

print "sorted?: #{nums}: "

atest(nums.sorted?)

#increasing

nums = nums.reverse

print "sorted?: #{nums}: "

atest(nums.sorted?)

Example

> array.rb

nums: [15, 3, 9, 3, 6, 2, 0, 1]

limited? to 1-15: false

limited? to 0-15: true

sorted?: [15, 3, 9, 3, 6, 2, 0, 1]: 0 = not sorted

sorted?: [0, 1, 2, 3, 3, 6, 9, 15]: 1 = increasing

sorted?: [15, 9, 6, 3, 3, 2, 1, 0]: -1 = decreasing

Question #3

3. Create a Ruby class triangle with initalizer, accessors, and member functions for computing the perimeter and the area of arbitrary triangles. Make also a member function test that checks sides a, b, and c and classifies the triangle as (1) equilateral, (2) isosceles, (3) scalene, (4) right, and (5) not a triangle. Right triangle can be either isosceles or scalene. Compute the perimeter and area only for valid triangles (verified by test). Show examples of the use of this class.

About my Code

Mutators and accessors are defined and the meat of the program rests inside of the test function. The test function is the corner stone of the entire program and runs through standard mathematical procedures to determine the type of triange.

Other standard formulas are easily applied to calculate the perimeter and area of the triangle.

Code

# CSc 600: Homework 5

# Ruby

#

# Keawa Rozet

# Homework 5: Question #3

class Triangle

def initialize(x,y,z)

@sideA = x

@sideB = y

@sideC = z

end

#mutators

def sideA=(x)

@sideA = x

end

def sideB=(y)

@sideB = y

end

def sideC=(z)

@sideC = z

end

#accessors

def sideA

@sideA

end

def sideB

@sideB

end

def sideC

@sideC

end

#returns type of triangle

def test

type = 5

#equilateral

if (@sideA == @sideB) and (@sideA == @sideC)

then type = 1

#isosceles

elsif (@sideA == @sideB) or (@sideA == @sideC) or (@sideB == @sideC)

then type = 2

#right

elsif (@sideA\*\*2 == (@sideB\*\*2+@sideC\*\*2)) or (@sideB\*\*2 == (@sideA\*\*2+@sideC\*\*2)) or (@sideC\*\*2 == (@sideA\*\*2+@sideB\*\*2))

then type = 4

#not a triangle

elsif (@sideA >= (@sideB+@sideC)) or (@sideB >= (@sideA+@sideC)) or (@sideC >= (@sideB+@sideA))

then type = 5

#scalene

else

type = 3

end

return type

end

def perimeter

if self.test < 5

return @sideA+@sideB+@sideC

else

put "This is not a triangle."

end

return

end

def area

if self.test < 5

perim = self.perimeter/2

return Math::sqrt(perim \* (perim-@sideA) \* (perim-@sideB) \* (perim-@sideC))

else

put "This is not a triangle."

end

return

end

end

#tells the type of triangle

def typeTest(type)

case type

when 1

puts "This is an equilateral triangle."

when 2

puts "This is an isosceles triangle."

when 3

puts "This is a scalene triangle."

when 4

puts "This is a right triangle."

when 5

puts "This is not a triangle."

end

end

#tests

tri = Triangle.new(9,12,15)

typeTest(tri.test)

puts "Side A = #{tri.sideA}, Side B = #{tri.sideB}, Side C = #{tri.sideC}, Perimeter: #{tri.perimeter}, Area: #{tri.area}\n"

tri.sideC = 6

typeTest(tri.test)

puts "Side A = #{tri.sideA}, Side B = #{tri.sideB}, Side C = #{tri.sideC}, Perimeter: #{tri.perimeter}, Area: #{tri.area}\n"

tri.sideB = 9

typeTest(tri.test)

puts "Side A = #{tri.sideA}, Side B = #{tri.sideB}, Side C = #{tri.sideC}, Perimeter: #{tri.perimeter}, Area: #{tri.area}\n"

tri.sideC = 9

typeTest(tri.test)

puts "Side A = #{tri.sideA}, Side B = #{tri.sideB}, Side C = #{tri.sideC}, Perimeter: #{tri.perimeter}, Area: #{tri.area}\n"

tri.sideA = 10

typeTest(tri.test)

puts "Side A = #{tri.sideA}, Side B = #{tri.sideB}, Side C = #{tri.sideC}, Perimeter: #{tri.perimeter}, Area: #{tri.area}\n"

Example

> ruby triangle.rb

This is a right triangle.

Side A = 9, Side B = 12, Side C = 15, Perimeter: 36, Area: 54.0

This is a scalene triangle.

Side A = 9, Side B = 12, Side C = 6, Perimeter: 27, Area: 19.078784028338912

This is an isosceles triangle.

Side A = 9, Side B = 9, Side C = 6, Perimeter: 24, Area: 25.45584412271571

This is an equilateral triangle.

Side A = 9, Side B = 9, Side C = 9, Perimeter: 27, Area: 28.844410203711913

This is an isosceles triangle.

Side A = 10, Side B = 9, Side C = 9, Perimeter: 28, Area: 37.416573867739416

Question #4

4. Create a Ruby class Sphere. Each sphere is characterized by the instance variable radius. For this class create the initializer and the following methods:

area – a method that returns the area of the sphere a = 4\*pi\*r^2

volume – a method that returns the volume of the sphere v = 4/3\*pi\*r^3

Create the class Ball that inherits properties from the class Sphere and adds a new instance variable color. Then create the class MyBall that inherits properties from the class Ball and adds a new instance variable owner. Write the method show that displays the instance variables of the class MyBall. Show sample applications of the class MyBall.

About my Code

The Sphere class uses the standard math formulas to calculate the area and volume of a sphere, given a radius upon creation.

The Ball class inherits from the Sphere class, meaning that it has access to the radius, area, and volume functions defined previousl in Sphere.

The MyBall class inherits from the Ball class, which means it also inherits from Ball’s super class – Sphere. MyBall has access to not only the color function defined in Ball, but also radius, area, and volume defined in Sphere. I show an example of this buy creating a MyBall object and calling both the area and volume functions from it.

Code

# CSc 600: Homework 5

# Ruby

#

# Keawa Rozet

# Homework 5: Question #4

class Sphere

def initialize(radius)

@radius = radius

end

#accessor

def radius

@radius

end

def area

return Math::PI \* 4 \* @radius\*\*2

end

def volume

return Math::PI \* 4/3 \* @radius\*\*3

end

end

class Ball < Sphere

def initialize(radius,color)

super(radius)

@color = color

end

#accessor

def color

@color

end

end

class MyBall < Ball

def initialize(radius,color,owner)

super(radius,color)

@owner = owner

end

#accessor

def owner

@owner

end

def show

puts "radius: #{@radius}, color: #{@color}, owner: #{@owner}"

end

end

#test

myb = MyBall.new(4,"red","Keawa")

myb.show

puts "area: #{myb.area}, volume: #{myb.volume}"

Example

> ruby sphere.rb

radius: 4, color: red, owner: Keawa

area: 201.06192982974676, volume: 268.082573106329