Shenliang Wang

12/10/2017

CSC 600

SECTION #1

HW#5

Ruby

1

#ConditionalStatements.rb

#loop-do

print "loop-do: "

x = 0

loop do

if x == 3

break

end

print "#{x = x + 1} "

end

#while-do

print "while-do: "

x = 0

while x < 7 do

print "#{x} "

x = x + 2

end

puts

#until-do

print "until-do: "

until x == 0 do

print "#{x} "

x = x - 1

end

puts

#while modifier

print "while modifier: "

x = 0

print "#{x = x + 1} " while x < 5

puts

#unto modifier

print "unto modifier: "

array = [1,2,3]

print array.pop.to\_s + " " until array.empty?

puts

#for-in

print "for-in: "

hash = {:Charmander => "fire", :Squirtle => "water", :Bulbasaur => "grass"}

for key,var in hash

print "#{key} => #{var}, "

end

puts

#upto

print "upto: "

1.upto(3){|counter| print "#{counter} "}

puts

#downto

print "downto: "

5.downto(1){|counter| print "#{counter} "}

puts

#times

print "times: "

3.times{print "Scary-Mary "}

puts

#each

print "each: "

array = [1,2,3]

array.each{|number| print "#{number} "}

puts

#map

print "map: "

sum = 0

(1..5).map{|y| print "#{(y\*y)} "}

puts

#step

print "step: "

1.step(2,0.2) do |y| print "#{y} " end

puts

#collect

print "collect: "

squares = [1,2,3].collect{|y| y\*\*2}

print squares

puts

#select

print "select: "

odds = (1..5).select{|y| y%2 == 1}

print odds

puts

#reject

print "reject: "

evens = (1..5).reject{|y| y%2 == 1}

print evens

puts

2．

#ArrayLimitedSorted

class Array

    #Checks all values in array meet amin <= num and num <= amax requirement

def limited?(amin,amax)

self.each do |num|

return false unless amin <= num && num <= amax

end

return true

end

#Checks if array is sorted either increasing or decreasing.

def sorted?

    self.each\_with\_index do |num, index|

        break unless num <= self[index+1] if index != self.length - 1

        return "+1" if index == self.length - 1

    end

    self.each\_with\_index do |num, index|

        break unless num >= self[index+1] if index != self.length - 1

        return "-1" if index == self.length - 1

    end

    return 0

end

end

#Print

array = [1,2,3,4,5]

puts "array: #{array} - array.limited?(0,10): #{array.limited?(0,10)},  array.sorted?: #{array.sorted?}"

array = [10,7,5,3,1]

puts "array: #{array} - array.limited?(5,11): #{array.limited?(5,11)},  array.sorted?: #{array.sorted?}"

array = [8, 3, 4 ,10, 5]

puts "array: #{array} - array.limited?(0,5): #{array.limited?(0,15)},  array.sorted?: #{array.sorted?}"

3．

#Triangle.rb

class Triangle

    #Getters and setters

    attr\_accessor :a, :b, :c

    def initialize(a,b,c)

        @a,@b,@c = a,b,c

    end

    #Test function classifies the trinagles as (1)equilateral, (2)isosceles, #(3)scalene, (4)right, and (5)not a triangle.

    def test

        if @a.to\_i != 0 && @b.to\_i != 0 && @c.to\_i != 0

            if  @a == @b && @a == @c

                return 1

            elsif @a\*\*2 == @b\*\*2 + @c\*\*2 || @b\*\*2 == @a\*\*2 + @c\*\*2 || @c\*\*2 == @b\*\*2 + @a\*\*2

                return 4

            elsif (@a == @b && @a != @c) || (@b == @c && @a != @c) || (@a == @c && @b != @c)

                return 2

            elsif @a != @b && @a != @c && @c != @b

                return 3

            end

        else

            return 5

        end

    end

    def perimiter

        return @a+@b+@c if self.test != 5

    end

    def area

        p = self.perimiter/2

        return Math.sqrt(p\*(p-@a)\*(p-@b)\*(p-@c)) if self.test != 5

    end

    def kind

        value = self.test

        if value ==  1

            return "is an equilateral triangle"

        elsif value == 2

            return "is an isosceles triangle"

        elsif value == 3

            return "is a scalene triangle"

        elsif value == 4

            return "is a right triangle"

        elsif value == 5

            return "is not a triangle"

        end

    end

end

#Print

t1 = Triangle.new(4,4,4)

print "t1: a = #{t1.a}, b = #{t1.b}, c = #{t1.c}, t1 #{t1.kind}\n"

puts "perimiter: #{t1.perimiter}, area: #{t1.area}"

t2 = Triangle.new(5,3,5)

print "t2: a = #{t2.a}, b = #{t2.b}, c = #{t2.c}, t2 #{t2.kind}\n"

puts "perimiter: #{t2.perimiter}, area: #{t2.area}"

t3 = Triangle.new(1,2,3)

print "t3: a = #{t3.a}, b = #{t3.b}, c = #{t3.c}, t3 #{t3.kind}\n"

puts "perimiter: #{t3.perimiter}, area: #{t3.area}"

t4 = Triangle.new(3,4,5)

print "t4: a = #{t4.a}, b = #{t4.b}, c = #{t4.c}, t4 #{t4.kind}\n"

puts "perimiter: #{t4.perimiter}, area: #{t4.area}"

t5 = Triangle.new("g",4,"C")

print "t5: a = #{t5.a}, b = #{t5.b}, c = #{t5.c}, t5 #{t5.kind}\n"

4.#Sphere.rb

class Sphere

    def initialize(radius)

        @radius = radius

    end

    def area

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

    end

    def volume

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

    end

end

class Ball < Sphere

    def initialize(radius,color)

        super(radius)

        @color = color

    end

end

class MyBall < Ball

    def initialize(radius,color,owner)

        super(radius,color)

        @owner = owner

    end

    def to\_s

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

    end

end

#Print

myball1 = MyBall.new(2, "Yellow", "Mike")

print "myball1: #{myball1.to\_s}, area: #{myball1.area}, volume: #{myball1.volume}"

puts

myball2 = MyBall.new(3, "Black", "Jimmy")

print "myball2: #{myball2.to\_s}, area: #{myball2.area}, volume: #{myball2.volume}"