**Jonathan Julian**

**CSc 600**

**12/10/19**

**Ruby HW**

**1.**

x = 1

puts "Loop testing"

loop do

puts"#{x}"

if x > 9 then

break

end

x += 1

end

puts "while testing"

while x > 0 do

puts "#{x}"

x -= 1

end

puts "until testing"

until x > 9 do

x += 1

puts "#{x}"

end

puts "for testing"

for i in (1 .. 10) do

puts"#{i}"

end

puts "upto testing"

x = 0

1.upto(10) do

x += 1

puts"#{x}"

end

puts "downto testing"

10.downto(1) do

x -= 1

puts"#{x}"

end

puts "times testing"

10.times do

x += 1

puts"#{x}"

end

puts "each testing"

array = [1,2,3,4,5,6,7,8,9,10]

array.each do |i|

puts"#{i}"

end

puts "map testing"

x = 1

array.map do |x|

puts "#{x \* 1}"

end

puts "step testing"

(1..10).step(1) do |x|

puts "#{x}"

end

puts "collect testing"

array.collect do |x|

puts "#{x}"

end

puts "select testing"

selectArray = array.select do |x|

x % 3 == 0

end

puts "#{selectArray}"

puts "reject testing"

rejectArray = array.reject { |x| x % 3 == 0}

puts"#{rejectArray}"

output:

Loop testing

1

2

3

4

5

6

7

8

9

10

while testing

10

9

8

7

6

5

4

3

2

1

until testing

1

2

3

4

5

6

7

8

9

10

for testing

1

2

3

4

5

6

7

8

9

10

upto testing

1

2

3

4

5

6

7

8

9

10

downto testing

9

8

7

6

5

4

3

2

1

0

times testing

1

2

3

4

5

6

7

8

9

10

each testing

1

2

3

4

5

6

7

8

9

10

map testing

1

2

3

4

5

6

7

8

9

10

step testing

1

2

3

4

5

6

7

8

9

10

collect testing

1

2

3

4

5

6

7

8

9

10

select testing

[3, 6, 9]

reject testing

[1, 2, 4, 5, 7, 8, 10]

**2.**

class Array

def limited?(amin, amax)

t = []

t = self.reject do |x| amin < x && x < amax

end

if t.empty?() then

return true

else return false

end

end

def sorted?

if (self.length() <= 1)

return 1

end

if (self[0] < self[1]) then

self.each do |x|

if x < self.length()-1 && x > self[x+1] then

return 0

end

end

return 1

end

if (self[0] > self[1]) then

for i in (0 .. self.length()-1) do

if i < self.length()-2 && self[i] < self[i+1] then

return 0

end

end

return -1

end

end

end

test = [1,2,3,4,5,6,7,8]

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

opposite = [9,8,7,6,5,4,3,2,1]

puts("#{test.limited?(0, 9)}")

puts("#{test.limited?(4,8)}")

puts("#{test.sorted?()}")

puts("#{unsorted.sorted?()}")

puts("#{opposite.sorted?()}")

output:

true

false

1

0

-1

**3.**

class Triangle

def initialize(a, b, c)

if a > b && a > c then

@hypotenuse = a

@side1 = b

@side2 = c

@right = self.checkRight()

elsif b > a && b > c then

@hypotenuse = b

@side1 = a

@side2 = c

@right = self.checkRight()

else

@hypotenuse = c

@side1 = a

@side2 = b

@right = self.checkRight()

end

end

def parimeter

@hypotenuse+@side1+@side2

end

def test

if (@side1 + @side2) < (@hypotenuse) then

return "Not a triangle"

elsif (@hypotenuse == @side1 && @side1 == @side2) then

return "Equlalateral"

elsif (@side1 == @side2) && @right then

return "Right Isoceles"

elsif (@side1 == @side2) then

return "Isoceles"

elsif @right then

return "Right Scalene"

else return "Scalene"

end

end

def checkRight

if self.test() == "Not a triangle" then

return false

end

angleA = Math.acos((@side1\*\*2 + @side2\*\*2 - @hypotenuse\*\*2)/(2\*@side1\*@side2))

angleB = Math.acos((@side1\*\*2 + @hypotenuse\*\*2 - @side2\*\*2)/(2\*@side1\*@hypotenuse))

angleC = 180 - (angleB + angleA)

if angleA == 90 || angleB == 90 || angleC == 90 then

return true

else return false

end

end

def area

#using heron's formula

if self.test() == "Not a triangle" then

return "Not a triangle"

end

s = (@side1 + @side2 + @hypotenuse)/2

Math.sqrt(s\*(s-@side1)\*(s-@side2)\*(s-@hypotenuse))

end

end

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

puts("#{triangle.parimeter()}")

puts("#{triangle.area()}")

puts(triangle.test())

triangle = Triangle.new(3,3,3)

puts("#{triangle.parimeter()}")

puts("#{triangle.area()}")

puts(triangle.test())

triangle = Triangle.new(4,5,6)

puts("#{triangle.parimeter()}")

puts("#{triangle.area()}")

puts(triangle.test())

triangle = Triangle.new(1,2,6)

puts("#{triangle.parimeter()}")

puts("#{triangle.area()}")

puts(triangle.test())

OUTPUT:

12

6.0

Scalene

9

2.0

Equlalateral

15

6.48074069840786

Scalene

9

Not a triangle

Not a triangle

**4.**

class Sphere

def initialize (radius)

@radius = radius

end

def area

4\*(@radius\*\*2)\*Math::PI

end

def volume

4\*(@radius\*\*3)\*Math::PI/3

end

attr\_accessor :radius

end

class Ball < Sphere

def initialize(radius,color)

super(radius)

@color = color

End

attr\_accessor :color

end

class MyBall < Ball

def initialize(radius, color, owner)

super(radius, color)

@owner = owner

end

attr\_accessor :owner

def show

puts("Radius is #{@radius}, Color is #{@color}, Owner is #{owner}")

end

end

ball = MyBall.new(3, "Blue", "Jonathan")

puts("#{ball.volume()}")

puts("#{ball.area()}")

ball.show()

otherBall = MyBall.new(16, "Red", "Jozo")

puts("#{otherBall.volume()}")

puts("#{otherBall.area()}")

otherBall.show()

OUTPUT:

113.09733552923255

113.09733552923255

Radius is 3, Color is Blue, Owner is Jonathan

17157.284678805056

3216.990877275948

Radius is 16, Color is Red, Owner is Jozo