

## Homework Problems:

### Problem 1:

```
# Sam Zandiasadabadi
# CSC 600-01
# HW 3.2: Ruby
# 05/18/2024
# Problem 1: Ruby demo program that illustrates the use of all main Ruby iterators
```

```
# writing a Ruby demo program for the 'loop' iterator
puts "loop:"
num = 0
loop do
  # displaying the current value of 'num' in the iteration
  puts num
  # increasing the value of num by 1
  num += 1
  # once the value of num equals 4, we end this demo
  if num == 4
    break
  end
end
```

```
# writing a Ruby demo program for the 'while' iterator
puts "\nwhile:"
num = 1
while num <= 4
  # while the value of num is less than or equal to 4, we print
  # the value of num within the current iteration
  puts num
  # increasing the value of num by 1
  num += 1
end
```

```
# writing a Ruby demo program for the 'until' iterator
puts "\nuntil:"
num = 2
# executing the loop 'until' the value of the iteration is bigger than or equal to 6
until num >= 6
  puts num
  num += 1
end
```

```
# writing a Ruby demo program for the 'for' iterator
puts "\nfor:"
# for every value in between numbers 3 and 6, we display the values
for num in 3..6 do
    puts num
end
```

```
# writing a Ruby demo program for the 'upto' iterator
puts "\nupto:"
# starting from number 4, and iterating upto number 7. we display all values
4.upto(7) do |num|
    puts num
end
```

```
# writing a Ruby demo program for the 'downto' iterator
puts "\ndownto:"
# starting from number 7, and iterating downto number 4. we display all values
7.downto(4) do |num|
    puts num
end
```

```
# writing a Ruby demo program for the 'times' iterator
puts "\ntimes:"
# running the loop until the value gets to 4
4.times do |num|
    puts num
end
```

```
# writing a Ruby demo program for the 'each' iterator
puts "\neach:"
num = 0
# writing sample labels to test the each iterator
numText = ['Number 0', 'Number 1', 'Number 2', 'Number 3', 'Number 4']
numText.each do |numText|
    # for every iteration in the array, we display the current values
    puts "Label: #{num} -> #{numText}"
    num += 1
end
```

```
# writing a Ruby demo program for the 'map' iterator
puts "\nmap:"
```

```

# defining a short array, and then adding 1 to each value in the array
puts [1, 2, 4, 6, 8].map{|num| num + 1}

# writing a Ruby demo program for the 'step' iterator
puts "\nstep:"
# starting from number 0, we iterate by 3 values until we reach number 15.
(0..15).step(3) do|num|
    puts num
end

# writing a Ruby demo program for the 'collect' iterator
puts "\ncollect:"
# defining a sample array
numArr = [1, 2, 4, 6, 8]
# using the 'collect' iterator to collect the values from the array and display them
collectValue = numArr.collect{|num| (num)}
puts collectValue

# writing a Ruby demo program for the 'select' and 'reject' iterator
puts "\nselect:"
# creating an array
numArr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# selecting and displaying all values smaller than 5
puts numArr.select {|num| num < 5}
puts "\nreject:"
# rejecting all values smaller than 5 and displaying the remaining ones
puts numArr.reject {|num| num < 5}

```

## Problem 2:

```

# Sam Zandiasadabadi
# CSC 600-01
# HW 3.2: Ruby
# 05/18/2024
# Problem 2: Ruby recognizer methods limited? and sorted? that expand the Ruby
class Array.
    # creating a class Array
    class Array
        # defining the 'limited?' method
        def limited?(amin,amax)
            # iterating through each 'num' element in the array that this function is called
            self.each do |num|
                # returning 'true' as long as 'num' is smaller than the largest index index

```

```

# value, and bigger than the smallest value
return true unless amax >= num && amin >= num
end
# returning false if other wise
return false
end

```

```

# defining the "sorted?" method
def sorted?
# initializing the value to 0
indexValue = 0
# iterating through each 'num' element in the array that this function is called
self.each do |num|
# if 'num' is less than or equal to the next element, and we have not reached
# the end of the array, we continue looping.
# However, if it is bigger, then we break the loop.
break unless num <= self[indexValue + 1] if indexValue != self.length - 1
# returning "+1" if the sequence is increasing
return "+1" if indexValue == self.length - 1
indexValue += 1
end

```

```

# resetting the value back to 0
indexValue = 0
self.each do |num|
# the exact opposite logic to the previous statement above
break unless num >= self[indexValue + 1] if indexValue != self.length - 1
# returning "-1" if the sequence is decreasing
return "-1" if indexValue == self.length - 1
indexValue += 1
end

```

```

# if the array is not sorted, then we return 0
return 0

```

```

end

```

```

end

```

```

# example 1 of this method
numArray = [1, 2, 4, 6, 8, 10]

```

```
puts "array: #{numArray}\narray.limited?: #{numArray.limited?(1, 10)}\narray.sorted?:  
#{numArray.sorted?}"
```

```
# example 2 of this method  
numArray = [10, 8, 6, 4, 2, 1]  
puts "\narray: #{numArray}\narray.limited?: #{numArray.limited?(10,  
1)}\narray.sorted?: #{numArray.sorted?}"
```

```
# example 3 of this method  
numArray = [10, 1, 8, 2, 6, 4]  
puts "\narray: #{numArray}\narray.limited?: #{numArray.limited?(10,  
4)}\narray.sorted?: #{numArray.sorted?}"
```

#### **Problem 4:**

```
# Sam Zandiasadabadi  
# CSC 600-01  
# HW 3.2: Ruby  
# 05/18/2024  
# Problem 4: Ruby class Sphere. Each sphere is characterized by the instance variable  
radius.  
# Ruby class Ball. Inherits properties from the class Sphere and adds a new instance  
variable color.  
# Ruby class MyBall. Inherits properties from the class Ball and adds a new  
instance variable owner
```

```
# creating the Sphere class  
class Sphere  
  # class constructor function  
  def initialize(radius)  
    @radius = radius  
  end  
  
  # creating this public function that returns the value of the radius  
  def radius
```

```

    @radius
  end

  # defining a function that calculates the area of the sphere
  def area
    # using the formula given in the instructions
    4 * (@radius ** 2) * Math::PI
  end

  # defining a function that calculates the volume of the sphere
  def volume
    # using the formula given in the instructions
    4 * (@radius ** 3) * Math::PI / 3.0
  end

  # defining the show method that displays the instance variables of the class
  def show
    puts "Radius: #{self.radius}"
  end

  # creating a method that displays the area and volume of the sphere class
  def areaAndVolume
    puts "Area: #{self.area}"
    puts "Volume: #{self.volume}"
  end

end

# creating the Ball class by inheriting from the Sphere class
class Ball < Sphere
  # class constructor function
  def initialize(radius, color)
    @radius = radius
    # adding the color variable
    @color = color
  end

  # creating this public function that returns the value of the color
  def color
    @color
  end
end

```

```

end

# defining the show method that displays the instance variables of the class
def show
  puts "Radius: #{self.radius}"
  puts "Color: #{self.color}"
end

end

# creating the MyBall class that inherits from the Ball class
class MyBall < Ball
  # class constructor function
  def initialize(radius, color, owner)
    @radius = radius
    @color = color
    # adding the owner variable
    @owner = owner
  end

  # creating this public function that returns the value of the owner
  def owner
    @owner
  end

  # defining the show method that displays the instance variables of the class
  def show
    puts "Radius: #{self.radius}"
    puts "Color: #{self.color}"
    puts "Owner: #{self.owner}"
  end

end

# creating a new Sphere object with radius size 2
sphere = Sphere.new(2)
puts "Sphere created"
# displaying the size of the radius
sphere.show
# displaying the area and the volume of the Sphere object

```

```
sphere.areaAndVolume
```

```
# creating a new Ball object with radius size 4 and color black
```

```
ball = Ball.new(4, "black")
```

```
puts "\nBall created"
```

```
# displaying the size of the radius
```

```
ball.show
```

```
# displaying the area and the volume of the Ball object
```

```
ball.areaAndVolume
```

```
# creating a new MyBall object with radius size 6, color green, and owner Sam
```

```
myball = MyBall.new(6, "green", "Sam")
```

```
puts "\nMyBall created"
```

```
# displaying the size of the radius
```

```
myball.show
```

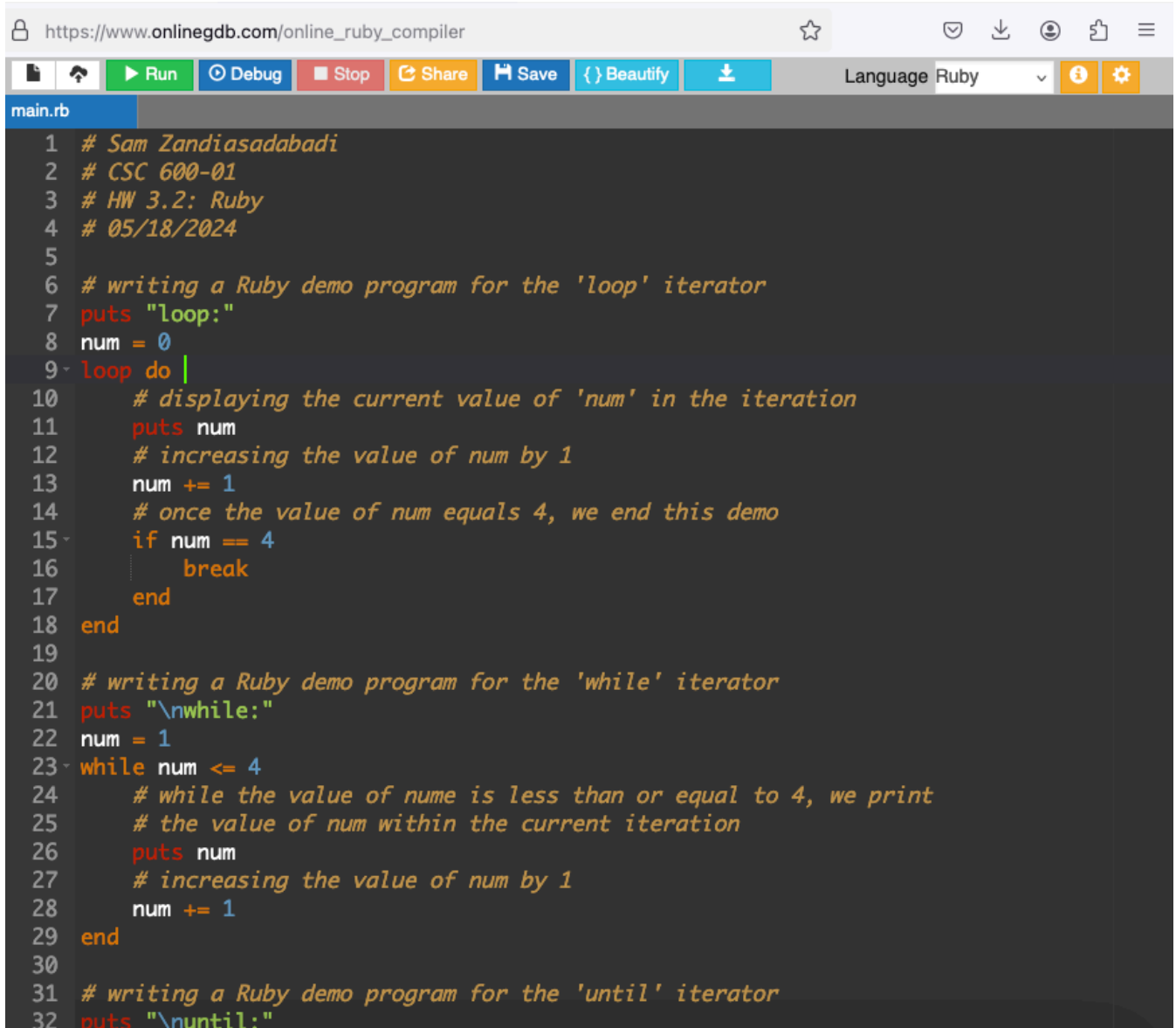
```
# displaying the area and the volume of the MyBall object
```

```
myball.areaAndVolume
```



## Screenshot of my Code:

### Problem 1:



The screenshot shows a web browser window with the URL [https://www.onlinegdb.com/online\\_ruby\\_compiler](https://www.onlinegdb.com/online_ruby_compiler). The interface includes a toolbar with buttons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The language is set to Ruby. The code is written in a dark-themed editor and consists of the following lines:

```
1 # Sam Zandiasadabadi
2 # CSC 600-01
3 # HW 3.2: Ruby
4 # 05/18/2024
5
6 # writing a Ruby demo program for the 'loop' iterator
7 puts "loop:"
8 num = 0
9 loop do |
10     # displaying the current value of 'num' in the iteration
11     puts num
12     # increasing the value of num by 1
13     num += 1
14     # once the value of num equals 4, we end this demo
15     if num == 4
16         break
17     end
18 end
19
20 # writing a Ruby demo program for the 'while' iterator
21 puts "\nwhile:"
22 num = 1
23 while num <= 4
24     # while the value of num is less than or equal to 4, we print
25     # the value of num within the current iteration
26     puts num
27     # increasing the value of num by 1
28     num += 1
29 end
30
31 # writing a Ruby demo program for the 'until' iterator
32 puts "\nuntil:"
```

https://www.onlinegdb.com/online\_ruby\_compiler

Run

Debug

Stop

Share

Save

{ } Beautify

Language Ruby

main.rb

```
33 num = 2
34 # executing the loop 'until' the value of the iteration is bigger than or equal to 6
35 until num >= 6
36     puts num
37     num += 1
38 end
39
40 # writing a Ruby demo program for the 'for' iterator
41 puts "\nfor:"
42 # for every value in between numbers 3 and 6, we display the values
43 for num in 3..6 do
44     puts num
45 end
46
47 # writing a Ruby demo program for the 'upto' iterator
48 puts "\nupto:"
49 # starting from number 4, and iterating upto number 7. we display all values
50 4.upto(7) do |num|
51     puts num
52 end
53
54 # writing a Ruby demo program for the 'downto' iterator
55 puts "\ndownto:"
56 # starting from number 7, and iterating downto number 4. we display all values
57 7.downto(4) do |num|
58     puts num
59 end
60
61 # writing a Ruby demo program for the 'times' iterator
62 puts "\ntimes:"
63 # running the loop until the value gets to 4
64 4.times do |num|
```

https://www.onlinegdb.com/online\_ruby\_compiler

RunDebugStopShareSaveBeautify

Language Ruby

main.rb

```
65     puts num
66 end
67
68 # writing a Ruby demo program for the 'each' iterator
69 puts "\neach:"
70 num = 0
71 # writing sample labels to test the each iterator
72 numText = ['Number 0', 'Number 1', 'Number 2', 'Number 3', 'Number 4']
73 numText.each do |numText|
74     # for every iteration in the array, we display the current values
75     puts "Label: #{num} -> #{numText}"
76     num += 1
77 end
78
79 # writing a Ruby demo program for the 'map' iterator
80 puts "\nmap:"
81 # defining a short array, and then adding 1 to each value in the array
82 puts [1, 2, 4, 6, 8].map{|num| num + 1}
83
84 # writing a Ruby demo program for the 'step' iterator
85 puts "\nstep:"
86 # starting from number 0, we iterate by 3 values until we reach number 15.
87 (0..15).step(3) do |num|
88     puts num
89 end
90
91 # writing a Ruby demo program for the 'collect' iterator
92 puts "\ncollect:"
93 # defining a sample array
94 numArr = [1, 2, 4, 6, 8]
95 # using the 'collect' iterator to collect the values from the array and display them
96
97 collectValue = numArr.collect{|num| (num)}
98 puts collectValue
99
100 # writing a Ruby demo program for the 'select' and 'reject' iterator
101 puts "\nselect:"
102 # creating an array
103 numArr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
104 # selecting and displaying all values smaller than 5
105 puts numArr.select {|num| num < 5}
106 # rejecting all values smaller than 5 and displaying the remaining ones
107 puts numArr.reject {|num| num < 5}
```

## Problem 2:

```
main.rb
1 # Sam Zandiasadabadi
2 # CSC 600-01
3 # HW 3.2: Ruby
4 # 05/18/2024
5 # Problem 2: Ruby recognizer methods limited? and sorted? that expand the Ruby class Array.
6
7 # creating a class Array
8 class Array
9   # defining the 'limited?' method
10  def limited?(amin,amax)
11    # iterating through each 'num' element in the array that this function is called
12    self.each do |num|
13      # returning 'true' as long as 'num' is smaller than the largestst index index
14      # value, and bigger than the smallest value
15      return true unless amax >= num && amin >= num
16    end
17    # returning false if other wise
18    return false
19  end
20
21  # defining the "sorted?" method
22  def sorted?
23    # initializing the value to 0
24    indexValue = 0
25    # iterating through each 'num' element in the array that this function is called
26    self.each do |num|
27      # if 'num' is less than or equal to the next element, and we have not reached
28      # the end of the array, we continue looping.
29      # However, if it is bigger, then we break the loop.
30      break unless num <= self[indexValue + 1] if indexValue != self.length - 1
31      # returning "+1" if the sequence is increasing
32      return "+1" if indexValue == self.length - 1
33      indexValue += 1
34    end
35
36    # resetting the value back to 0
37    indexValue = 0
38    self.each do |num|
39      # the exact opposite logic to the previous statement above
40      break unless num >= self[indexValue + 1] if indexValue != self.length - 1
41      # returning "-1" if the sequence is decreasing
42      return "-1" if indexValue == self.length - 1
43      indexValue += 1
44    end
45
46    # if the array is not sorted, then we return 0
47    return 0
48  end
49 end
50
51 # example 1 of this method
52 numArray = [1, 2, 4, 6, 8, 10]
53 puts "array: #{numArray}\narray.limited?: #{numArray.limited?(1, 10)}\narray.sorted?: #{numArray.sorted?}"
54
55 # example 2 of this method
56 numArray = [10, 8, 6, 4, 2, 1]
57 puts "\narray: #{numArray}\narray.limited?: #{numArray.limited?(10, 1)}\narray.sorted?: #{numArray.sorted?}"
58
59 # example 3 of this method
60 numArray = [10, 1, 8, 2, 6, 4]
61 puts "\narray: #{numArray}\narray.limited?: #{numArray.limited?(10, 4)}\narray.sorted?: #{numArray.sorted?}"
```

## Problem 4:

```
main.rb
1 # Sam Zandiasadabadi
2 # CSC 600-01
3 # HW 3.2: Ruby
4 # 05/18/2024
5 # Problem 4: Ruby class Sphere. Each sphere is characterized by the instance variable radius.
6 #           Ruby class Ball. Inherits properties from the class Sphere and adds a new instance variable color.
7 #           Ruby class MyBall. Inherits properties from the class Ball and adds a new instance variable owner
8
9
10 # creating the Sphere class
11 class Sphere
12   # class constructor function
13   def initialize(radius)
14     @radius = radius
15   end
16
17   # creating this public function that returns the value of the radius
18   def radius
19     @radius
20   end
21
22   # defining a function that calculates the area of the sphere
23   def area
24     # using the formula given in the instructions
25     4 * (@radius ** 2) * Math::PI
26   end
27
28   # defining a function that calculates the volume of the sphere
29   def volume
30     # using the formula given in the instructions
31     4 * (@radius ** 3) * Math::PI / 3.0
32   end
33
34   # defining the show method that displays the instance variables of the class
35   def show
36     puts "Radius: #{self.radius}"
37   end
38
39   # creating a method that displays the area and volume of the sphere class
40   def areaAndVolume
41     puts "Area: #{self.area}"
42     puts "Volume: #{self.volume}"
43   end
44 end
45
46 # creating the Ball class by inheriting from the Sphere class
47 class Ball < Sphere
48   # class constructor function
49   def initialize(radius, color)
50     @radius = radius
51     # adding the color variable
52     @color = color
53   end
54 end
55
56 # creating this public function that returns the value of the color
57 def color
58   @color
59 end
60
61 # defining the show method that displays the instance variables of the class
62 def show
63   puts "Radius: #{self.radius}"
64   puts "Color: #{self.color}"
65 end
```

```

62-   def show
63-       puts "Radius: #{self.radius}"
64-       puts "Color: #{self.color}"
65-   end
66-
67- end
68-
69- # creating the MyBall class that inherits from the Ball class
70- class MyBall < Ball
71-     # class constructor function
72-     def initialize(radius, color, owner)
73-         @radius = radius
74-         @color = color
75-         # adding the owner variable
76-         @owner = owner
77-     end
78-
79-     # creating this public function that returns the value of the owner
80-     def owner
81-         @owner
82-     end
83-
84-     # defining the show method that displays the instance variables of the class
85-     def show
86-         puts "Radius: #{self.radius}"
87-         puts "Color: #{self.color}"
88-         puts "Owner: #{self.owner}"
89-     end
90-
91- end
92-
93- # creating a new Sphere object with radius size 2

```

```

93- # creating a new Sphere object with radius size 2
94- sphere = Sphere.new(2)
95- puts "Sphere created"
96- # displaying the size of the radius
97- sphere.show
98- # displaying the area and the volume of the Sphere object
99- sphere.areaAndVolume
100-
101- # creating a new Ball object with radius size 4 and color black
102- ball = Ball.new(4, "black")
103- puts "\nBall created"
104- # displaying the size of the radius
105- ball.show
106- # displaying the area and the volume of the Ball object
107- ball.areaAndVolume
108-
109- # creating a new MyBall object with radius size 6, color green, and owner Sam
110- myball = MyBall.new(6, "green", "Sam")
111- puts "\nMyBall created"
112- # displaying the size of the radius
113- myball.show
114- # displaying the area and the volume of the MyBall object
115- myball.areaAndVolume
116-
117-

```

## Output:

### Problem 1:

```
https://www.onlinegdb.com/online_ruby_compiler
loop:
0
1
2
3
while:
1
2
3
4
until:
2
3
4
5
for:
3
4
5
6
upto:
4
5
6
7
downto:
7
6
5
4
times:
0
1
2
3
```

```
each:
Label: 0 -> Number 0
Label: 1 -> Number 1
Label: 2 -> Number 2
Label: 3 -> Number 3
Label: 4 -> Number 4

map:
2
3
5
7
9

step:
0
3
6
9
12
15

collect:
1
2
4
6
8

select:
0
1
2
3
4

reject:
5
6
7
8
9

...Program finished with exit code 0
Press ENTER to exit console.
```

## Problem 2:

```
array: [1, 2, 4, 6, 8, 10]
array.limited?: true
array.sorted?: +1

array: [10, 8, 6, 4, 2, 1]
array.limited?: true
array.sorted?: -1

array: [10, 1, 8, 2, 6, 4]
array.limited?: true
array.sorted?: 0

...Program finished with exit code 0
Press ENTER to exit console.
```



#### Problem 4:

```
Sphere created  
Radius: 2  
Area: 50.26548245743669  
Volume: 33.510321638291124
```

```
Ball created  
Radius: 4  
Color: black  
Area: 201.06192982974676  
Volume: 268.082573106329
```

```
MyBall created  
Radius: 6  
Color: white  
Owner: Sam  
Area: 452.3893421169302  
Volume: 904.7786842338603
```

```
...Program finished with exit code 0  
Press ENTER to exit console.□
```