

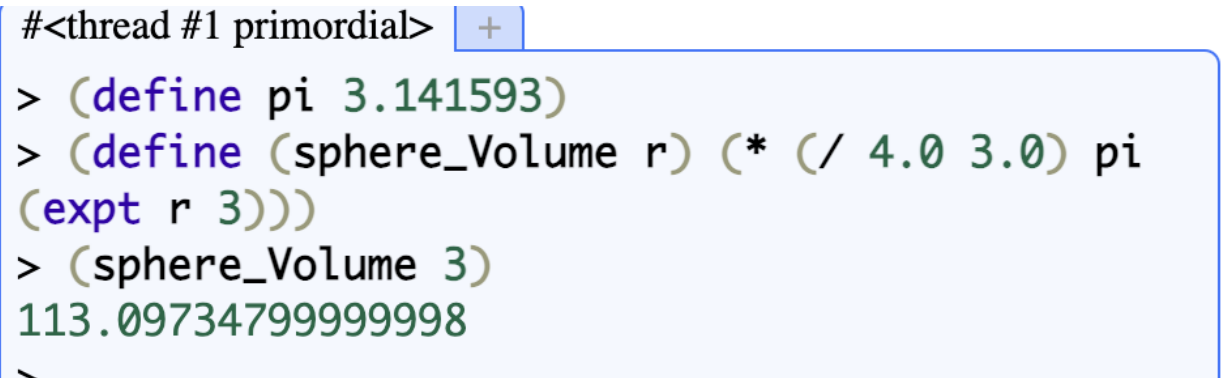
Assignment #4 Functional Programming and Scripting Languages

Question 1: Given the volume formula of a sphere

a. Scheme Sources code

```
> (define pi 3.141593)
> (define (sphere_Volume r) (* (/ 4.0 3.0) pi (expt r 3)))
> (sphere_Volume 3)
```

Image of the output generated when the radius is set to 3.



```
#<thread #1 primordial> +
> (define pi 3.141593)
> (define (sphere_Volume r) (* (/ 4.0 3.0) pi
  (expt r 3)))
> (sphere_Volume 3)
113.09734799999998
~
```

b. Ruby Source code

```
def sphere_Volume()
  pi = 3.141593
  puts "Radius:"
  radius=gets.chomp.to_f
  volume_equation = (4.0 / 3.0) * pi * (radius ** 3)
```

```

    puts "Sphere Volume:  #{volume_equation}"
    return volume_equation
end
volume= sphere_Volume()

```

Image of the output generated when the radius is set to 3.

The screenshot shows a Ruby IDE with a file named 'HelloWorld.rb'. The source code on the left is as follows:

```

1 def sphere_Volume()
2   pi = 3.141593
3   puts "Radius:"
4   radius=gets.chomp.to_f
5   volume_equation = (4.0 / 3.0) * pi * (radius ** 3)
6   puts "Sphere Volume:  #{volume_equation}"
7   return volume_equation
8 end
9 volume= sphere_Volume()
10
11

```

On the right, the 'STDIN' input is '3'. The 'Output' section shows the following text:

```

Radius:
Sphere Volume:  113.09734799999998

```

Question 2:

a. Recursion Source code and the Output

```

def factorial(n)
  if n == 0
    return 1
  else
    #Recursion happens here at factorial(n-1)
    return n * factorial(n - 1)
  end
end

result= factorial(5)
puts "#{result}"

```

HelloWorld.rb	
<pre>1 def factorial(n) 2 if n == 0 3 return 1 4 else 5 <i>#Recurrsion happens here at factorial(n-1)</i> 6 return n * factorial(n - 1) 7 end 8 end 9 10 result= factorial(5) 11 puts "#{result}" 12</pre>	<p>STDIN</p> <p>Input for the program (Option)</p> <hr/> <p>Output:</p> <p>120</p>

b. Loop Source code and the Output

```
def factorial_loop(n)
  if n == 0
    return 1
  end

  result = 1
  i = n

  while i > 1
    result *= i
    i -= 1
  end

  return result
end

result = factorial_loop(1)
puts result
```

HelloWorld.rb



```
1 def factorial_loop(n)
2   if n == 0
3     return 1
4   end
5
6   result = 1
7   i = n
8
9   while i > 1
10    result *= i
11    i -= 1
12  end
13
14  return result
15 end
16
17 result = factorial_loop(5)
18 puts result
```

STDIN

Input for the program (Optional)

Output:

120

c. Class

```
class FindFactorial
  def initialize(factorial_number)
    @factorial_number = factorial_number
  end

  def factorial()
    n = @factorial_number
    if n == 0
      return 1
    end

    result = 1
    i = n

    while i > 1
      result *= i
    end
  end
end
```

```
        i -= 1
    end

    return result
end
end

find = FindFactorial.new(5)
result = find.factorial
puts result
```

HelloWorld.rb



```
1 class FindFactorial
2   def initialize(factorial_number)
3     @factorial_number = factorial_number
4   end
5
6   def factorial()
7     n = @factorial_number
8     if n == 0
9       return 1
10    end
11
12    result = 1
13    i = n
14
15    while i > 1
16      result *= i
17      i -= 1
18    end
19
20    return result
21  end
22 end
23
24 find = FindFactorial.new(5)
25 result = find.factorial
26 puts result
27
```

STDIN

Output:

120

Question 3:

Scripting Language

a. Error handling

Scripting languages have inbuilt datasets that can catch errors. There are also libraries that handle missing values and incorrect formats. Functional programming is also good for catching errors. In functional programming, the code is executed sequentially, meaning if you encounter an error, you must fix it before proceeding to the next line of code. Object-oriented programming has embedded error handling, but it can make the code harder to read.

b. Suitability for Handling Complex Data Structures

Scripting languages excel at handling complex data structures because they have built-in data types such as dictionaries and lists. Additionally, some scripting languages are dynamically typed, meaning they can mix different data types within data structures. Functional programming is satisfactory, although it can handle complex data structures; it doesn't have built-in libraries to read complex datasets. Object-oriented programming is the best when it comes to handling complex data structures because of polymorphism, but it increases the size of the dataset.

c. Performance

Scripting languages are slow when it comes to performance, but they have built-in libraries that optimize the background, resulting in better performance. Functional programming varies in performance depending on the dataset given. Object-oriented programming is the best regarding performance because it can handle any dataset due to the vast libraries that address all performance issues.