Homework 5 Solutions

Problem 1 (Roots)

In [2]:
```
import scala.math.abs
import scala.math.pow

val tolerance = 0.000001

def isCloseEnough(x: Double, y: Double) =
    abs((x-y)/x) < tolerance

def fixpoint(f: Double => Double)(firstGuess: Double) = {
    def iterate(guess: Double): Double = {
        val next = f(guess)
        println(next)
        if (isCloseEnough(guess, next)) next
        else iterate(next)
    }
    iterate(firstGuess)
}

def roota = fixpoint(x => pow((x + 10),0.25))(4.0)

roota
```

```
1.9343364202676694
1.8586583582639158
1.8557047925596433
1.8555892344194143
1.8555847127729057
1.8555845358458216
```

Out[2]:

```
import scala.math.abs

import scala.math.pow


tolerance: Double = 1.0E-6
defined function isCloseEnough
defined function fixpoint
defined function roota
res1_6: Double = 1.8555845358458216
```

In [3]:

```
import scala.math.abs
import scala.math.cos
import scala.math.exp

val tolerance = 0.000001

def isCloseEnough(x: Double, y: Double) =
    abs((x-y)/x) < tolerance

def fixpoint(f: Double => Double)(firstGuess: Double) = {
    def iterate(guess: Double): Double = {
        val next = f(guess)
        println(next)
        if (isCloseEnough(guess, next)) next
        else iterate(next)
    }
    iterate(firstGuess)
}

def rootb = fixpoint(x => cos(x)/exp(x))(2.0)

rootb
```

```
-0.056319349992127885
1.0562581084452713
0.17114150288044905
0.8303912842995452
0.29403809072061776
0.7132630725019496
0.370585293255405 5
0.6434674726335474
```

```
0.4203843791794768
0.5996086367572037
0.45325238263448697
0.5713839656149187
0.47503572163316676
0.5530079396889118
0.48948000748056564
0.5409714756715229
0.4990518813409288
0.533061356741054
0.50538966692802
0.5278531325633876
0.5095830130705244
0.5244200424249766
0.5123559467103711
0.5221555019604824
0.5141888586743251
0.5206611171535565
0.5154000714046607
0.5196746919152099
0.5162003015423904
0.519023449301237
0.5167289322743666
0.5185934463932321
0.5170781141797129
0.5183095024605553
0.5173087493825228
0.5181219964577776
0.5174610783659128
0.5179981704675224
0.5175616853828159
0.5179163960059279
0.5176281310353136
0.5178623915403935
0.5176720144035084
0.5178267262576728
0.5177009965192696
0.5178031722765886
0.5177201372305206
0.5177876167497917
0.5177327783238355
0.5177773435370843
0.5177411268586974
0.5177705588690933
0.5177466404589214
0.5177660781117629
0.5177502817875248
0.5177631189099772
0.5177526866165311
0.5177611645797174
```

```
0.5177542748280883
0.5177598738911071
0.5177553237239807
0.5177590214878617
0.5177560164417679
0.5177584585392209
0.5177564739303324
0.5177580867536788
0.5177567760674712
0.5177578412170739
0.5177569756065482
0.5177576790584633
0.5177571073872425
0.5177575719647951
```

Out[3]:
```
import scala.math.abs

import scala.math.cos

import scala.math.exp


tolerance: Double = 1.0E-6
defined function isCloseEnough
defined function fixpoint
defined function rootb
res2_7: Double = 0.5177575719647951
```

Problem 2 (Word Value)

In [4]:
```
def ord(c: Char): Int =
  c.toInt - 'a'.toInt + 1

def wordValue(s: String): Int = {
    if (s == "") 0
    else ord(s.toLowerCase().charAt(0)) + wordValue(s.toLowerCase().su
bstring(1))
}
wordValue("Attitude")
```

Out[4]:
```
defined function ord
defined function wordValue
res3_2: Int = 100
```

Problem 3 (HOF)

In [5]:
```
def compose(f:Int=>Int, g: Int=>Int): Int=>Int =
  x => f(g(x))

def repeated(f:Int=>Int,n:Int): Int=>Int = {
    if (n == 1) f
    else compose(f,repeated(f,n-1))
}

compose(x=>x*x, x=>x+1)(6)
repeated(x=>3*x,4)(8)
```

Out[5]:
```
defined function compose
defined function repeated
res4_2: Int = 49
res4_3: Int = 648
```

Problem 4 (Rational)

In [6]:

```
import scala.math.abs

type Rational = (Int,Int)

def gcd(a: Int, b: Int): Int =
    if (b == 0) a else gcd(b, a%b)

def makeRational(n:Int, d:Int): Rational = {
    if (d == 0)
        throw new Exception("Divide by 0")
    val g = gcd(abs(n),abs(d))
    if (((n >= 0) && (d >= 0)) || ((n < 0) && (d < 0)))
        (abs(n)/g,abs(d)/g)
    else
        (-abs(n)/g,abs(d)/g)
}

def numer(r:Rational):Int =
  r._1

def denom(r:Rational):Int =
  r._2

def addRational(r1:Rational,r2:Rational): Rational =
  makeRational(numer(r1)*denom(r2)+denom(r1)*numer(r2),denom(r1)*denom
(r2))

def subRational(r1:Rational,r2:Rational): Rational =
  makeRational(numer(r1)*denom(r2)-denom(r1)*numer(r2),denom(r1)*denom
(r2))

def mulRational(r1:Rational,r2:Rational): Rational =
  makeRational(numer(r1)*numer(r2),denom(r1)*denom(r2))

def divRational(r1:Rational,r2:Rational): Rational =
  makeRational(numer(r1)*denom(r2),denom(r1)*numer(r2))

def equalRational(r1:Rational,r2:Rational): Boolean =
  numer(r1) * denom(r2) == numer(r2) * denom(r1)

def to_string(r: Rational): String =
  numer(r).toString + "/" + denom(r).toString

//val r0 = makeRational(1,0)
val r1 = makeRational(-1,3)
val r2 = makeRational(-1,-3)
println(to_string(addRational(r1,r2)))
```

```
0/1
```

```
Out[6]: import scala.math.abs


        defined type Rational
        defined function gcd
        defined function makeRational
        defined function numer
        defined function denom
        defined function addRational
        defined function subRational
        defined function mulRational
        defined function divRational
        defined function equalRational
        defined function to_string
        r1: (Int, Int) = (-1, 3)
        r2: (Int, Int) = (1, 3)
```

Problem 5 (Convert)

In [7]:
```
def convertNum2Binary(num: Int): String =
    if (num <= 1) num.toString
    else convertNum2Binary(num/2)++(num%2).toString

def convertFraction2Binary(num: Double): String = {
    def helper(num: Double, res: String): String = {
      if (res.length >= 23) res
      else
        if (num*2 == 1)
          res++"1"
        else if (num*2 < 1)
          helper(num*2,res++"0")
        else
          helper(num*2-1.0,res++"1")
    }
    "."++helper(num,"")
}

println(convertNum2Binary(100))
println(convertFraction2Binary(0.375))
println(convertFraction2Binary(0.8))
```

```
1100100
.011
.11001100110011001100110
```

Out[7]:
```
defined function convertNum2Binary
defined function convertFraction2Binary
```