19CSE313

Principles of Programming Languages

Lab 4

S Abhishek

AM.EN.U4CSE19147

Determine the value of the following expressions.

Try each expression in the terminal.

1. let x = pi / 2 in sin x ** 2
2. let x = 1 in if x > 0 then x else (negate x)
3. if True then let x = 2 in x ^ 4 else 0
4. negate (let { x = 1; y = 2 } in x + y)
5. let x = 1 in let y = 2 in x + y
6. let x = 1 in let x = 2 in x
7. let x = 1 in let y = 2 in let x = 0 in x + y
8. let y = 7 in let x = 3 in 5 + (let x = 2 in x + y) * x

```
◔_◔ a3x3k ◔_◔ → ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> let x = pi / 2 in sin x ** 2
1.0
Prelude> let x = 1 in if x > 0 then x else (negate x)
1
Prelude> let x = -1 in if x > 0 then x else (negate x)
1
Prelude> if True then let x = 2 in x ^ 4 else 0
16
Prelude> negate (let { x = 1; y = 2 }  in x + y)
-3
Prelude> let x = 1 in let y = 2 in x + y
3
Prelude> let x = 1 in let x = 2 in x
2
Prelude> let x = 1 in let y = 2 in let x = 0 in x + y
2
Prelude> let y = 7 in let x = 3 in 5 + (let x = 2 in x + y ) * x
32
```

1 - Define a function to find the largest of 3 numbers using if expression.

```
max3 :: (RealFloat x) => x -> x -> x -> x

max3 x y z =

 if (x >= y) && (x >= z) then x
 else if (y >= x) && (y >= z) then y
 else z
```

```
*Main> max3 2 4 5
5.0
*Main> max3 4.51 4.50 4.52
4.52
*Main> max3 4.5 4 5
5.0
*Main> max3 100 9 200
200.0
```

2 - Define a function of type : Int -> String which reads a number and returns whether "even" or "odd".

```
evenodd :: Int -> String

evenodd x =
 if x `mod` 2 == 0 then "Even"
 else "Odd"
```

```
*Main> evenodd 10
"Even"
*Main> evenodd 13
"Odd"
*Main> evenodd 1
"Odd"
*Main> evenodd 5
"Odd"
*Main> evenodd 5100
"Even"
```

3 - Using Guards, determine the largest of two numbers.

```
maxof2 :: (RealFloat x) => x -> x -> x
maxof2 x y | x >= y = x
 | otherwise = y
```

```
*Main> maxof2 2 5
5.0
*Main> maxof2 0 1
1.0
*Main> maxof2 101 1000
1000.0
```

4 - Define a function distance to find the distance between two points in a xy-plane.

$$PQ = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Let P = (x1, y1) and Q = (x2, y2).

```
dist :: (RealFloat x) => x -> x -> x -> x -> x

dist x1 y1 x2 y2 = sqrt((a)^2 + (b)^2) where
 a = x2 - x1
 b = y2 - y1
```

```
*Main> dist 5 4 6 9
5.0990195135927845
*Main> dist 2 4 1 9
5.0990195135927845
*Main> dist 9 2 8 0
2.23606797749979
*Main> dist 4 0 7 0
3.0
```

5 - Define the function min and max which work with values of arbitrary type, so long as this type is an instance of the Ord class.

- Check this function, by passing different type of values, like int, float, char, string.

```
min1 :: (Ord a) => a -> a -> a

min1 x y | x < y = x
 | otherwise = y


max1 :: (Ord a) => a -> a -> a

max1 x y | x > y = x
 | otherwise = y
```

```
*Main> min1 2 5
2
*Main> min1 2.5 4
2.5
*Main> min1 1.0 1.1
1.0
*Main>
*Main> max1 1.0 1.1
1.1
*Main> max1 2.5 4
4.0
*Main> max1 2 10
10
```

6 - Define a function divides, divides :: Int -> Int -> Bool which, verifies whether the first argument divides the second one.

- Define this function using if expression, guarded expression and multiple definition using pattern matching.

```
divides :: Int -> Int -> Bool
divides 0 _ = False
divides x y = y `mod` x == 0

divide :: Int -> Int -> Bool
divide x y = if x == 0 then False
 else if y `mod` x == 0 then True
 else False

divi :: Int -> Int -> Bool
divi x y | x == 0 = False
 | y `mod` x == 0 = True
 | otherwise = False
```

```
*Main> 2 `divides` 3
False
*Main> 0 `divides` 3
False
*Main> 2 `divides` 4
True
*Main> 10 `divides` 100
True
```

```
*Main> 2 `divide` 3
False
*Main> 0 `divide` 3
False
*Main> 2 `divide` 4
True
*Main> 10 `divide` 100
True
```

```
*Main> 2 `divi` 3
False
*Main> 0 `divi` 3
False
*Main> 2 `divi` 4
True
*Main> 10 `divi` 100
True
```

7 - Implement a function in Haskell for the following mathematic function defined as, [use pattern matching]

$$f(x) = \begin{cases} 7 & \text{if } x = 0 \\ 3x^2 - 2 & \text{otherwise} \end{cases}$$

```
fun :: (Integral a) => a -> a

fun 0 = 7

fun x = ( 3 * x * x ) - 2
```

```
*Main> fun 0
7
*Main> fun 2
10
*Main> fun 5
73
*Main> fun 10
298
*Main> fun 12
430
*Main> fun 1
1
```

8 - Define a function to implement Stirling's formula

```
stirling :: Int -> Float

stirling x = sqrt ( 2 * pi * a ) * ( (a / exp 1) ** a ) where
 a :: Float
 a = fromIntegral x
```

```
*Main> stirling 2
1.9190046
*Main> stirling 5
118.01921
*Main> stirling 10
3598697.8
*Main> stirling 3
5.836211
```

9 - Define a function noOfSol of type :: Int -> Int -> Int -> String, to find the number of solution of a quadratic equation.

```
noOfSol :: Int -> Int -> Int -> String

noOfSol a b c | sol > 0 = "2 Solutions"
 | sol == 0 = "1 Solution"
 | otherwise = "No Solution" where
 sol = (b * b) - (4 * a * c)
```

```
*Main> noOfSol 2 5 2
"2 Solutions"
*Main> noOfSol 2 2 3
"No Solution"
*Main> noOfSol 1 4 5
"No Solution"
*Main> noOfSol 10 5 20
"No Solution"
```

10 - Define a function rootsOfQuadraticEqu of appropriate type, to find the two roots of a Quadratic equation.

- Given a, b and c, find the roots x1 and x2.

```
rootsOfQuadraticEqu :: (Float, Float, Float) -> (Float, Float)

rootsOfQuadraticEqu (a, b, c) = (x, y) where
 x = eq + sqrt root / ( 2 * a )
 y = eq - sqrt root / ( 2 * a )
 eq = - b / (2 * a)
 root = ( b * b ) - ( 4 * a * c )
```

```
*Main> rootsOfQuadraticEqu (1, 2, 1)
(-1.0,-1.0)
*Main> rootsOfQuadraticEqu (1, 10, -24)
(2.0,-12.0)
*Main> rootsOfQuadraticEqu (1, 4, 2)
(-0.58578646,-3.4142137)
```

Thankyou!!