1. Write a function in Haskell to evaluate Area of a triangle

Code:

```haskell
-- Author: Ajaykumar

areaOfTriangle :: Float -> Float -> Float
areaOfTriangle x y = x * y / 2

main :: IO ()
main = do
  putStr "Height = 3, Base = 4\nArea Of Triangle = "
  print (truncate (areaOfTriangle 2 3))

```

Output:

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_4> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l area_of_triangle.hs
[1 of 1] Compiling Main             ( area_of_triangle.hs, interpreted )
Ok, one module loaded.
ghci> main
Height = 3, Base = 4
Area Of Triangle = 3
ghci>
```

2. Write a function in Haskell to evaluate Area of a Circle

Code:

```haskell
-- Author: Ajaykumar

areaOfCircle :: Float -> Float
areaOfCircle r = pi * r**2

main :: IO ()
main = do
  putStr "Radius = 3\nArea Of Circle = "
  print (areaOfCircle 3)

```

Output:

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_4> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l area_of_circle.hs
[1 of 1] Compiling Main             ( area_of_circle.hs, interpreted )
Ok, one module loaded.
ghci> main
Radius = 3
Area Of Circle = 28.274334
ghci>
```

3. Write a function in Haskell to Implement XOR operations

Code:

```haskell
-- Author: Ajaykumar

xor :: Bool -> Bool -> Bool
xor x y | x == y = False
xor x y | x /= y = True

main :: IO ()
main = do
  putStr "xor False True = "
  print (xor False True)

  putStr "xor False False = "
  print (xor False False)

```

Output:

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_4> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l xor.hs
[1 of 1] Compiling Main             ( xor.hs, interpreted )
Ok, one module loaded.
ghci> main
xor False True = True
xor False False = False
ghci>
```

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> zipWith(+) [4,2,5,6][2,6,2,3]
[6,8,7,9]
ghci> zipWith max [6, 3, 2, 1] [7, 3, 1, 5]
[7,3,2,5]
ghci> zipWith (++)["foo", "bar", "baz"]["fighters", "hoppers", "aldrin"]
["foofighters","barhoppers","bazaldrin"]
ghci> zipWith(*)(replicate 5 2)[1..]
[2,4,6,8,10]
ghci> map(map(^2))[[1,2], [3,4,5,6], [7,8]]
[[1,4],[9,16,25,36],[49,64]]
ghci> map(++"!")["BIFF", "BANG", "POW"]
["BIFF!","BANG!","POW!"]
ghci> map(replicate 3)[3..6]
[[3,3,3],[4,4,4],[5,5,5],[6,6,6]]
ghci>
```

5. Write a function flip that simply takes a function and returns a function that is like our original function.

Code:

```haskell
-- Author: Ajaykumar Nadar

flipFunction :: (a -> b -> c) -> b -> a -> c
flipFunction f x y = f y x

main :: IO ()
main = do
  putStr "5 - 3 = "
  print (flipFunction (-) 5 3)
  putStr "5 / 2 = "
  print (flipFunction (/) 5 2)
```

Output:

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\lab\Exp_4> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l flip.hs
[1 of 1] Compiling Main             ( flip.hs, interpreted )
Ok, one module loaded.
ghci> main
5 - 3 = -2
5 / 2 = 0.4
ghci>
```

6. Write a function dividebyTen that divides the argument by 10

Code:

```haskell
-- Author: Ajaykumar Nadar

divideByTen :: Float -> Float
divideByTen x = (x / 10)

main :: IO ()
main = do
  putStr "25 / 10 = "
  print (divideByTen 25)

```

Output:

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_4> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l divideByTen.hs
[1 of 1] Compiling Main             ( divideByTen.hs, interpreted )
Ok, one module loaded.
ghci> main
25 / 10 = 2.5
ghci>
```