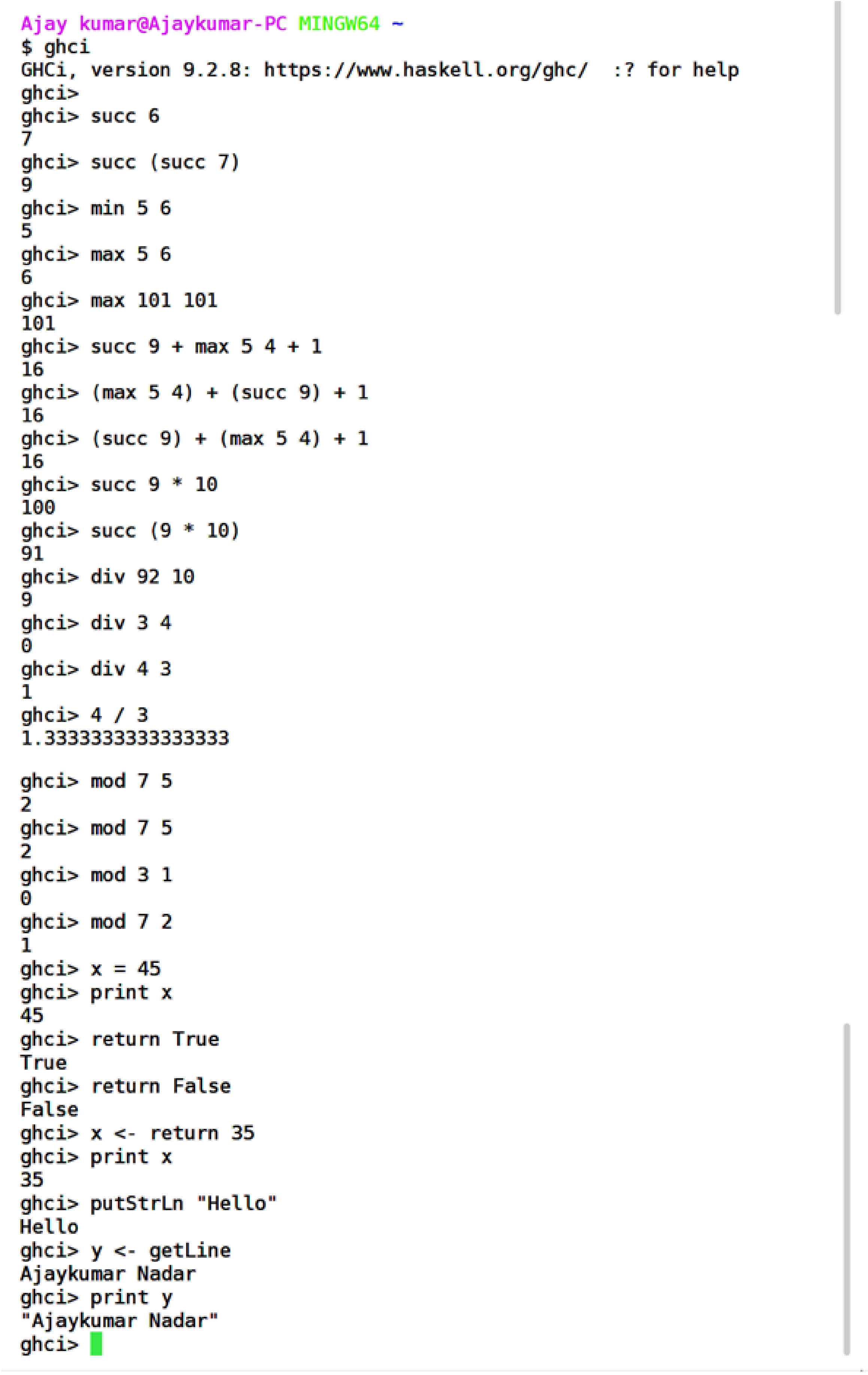
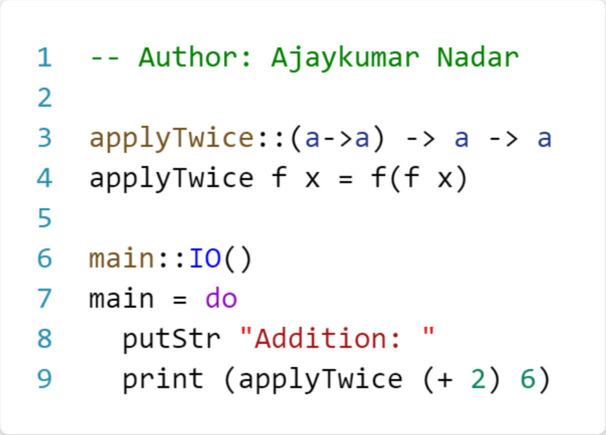


```
Ajay kumar@Ajaykumar-PC MINGW64 ~
$ ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci>
ghci> succ 6
7
ghci> succ (succ 7)
9
ghci> min 5 6
5
ghci> max 5 6
6
ghci> max 101 101
101
ghci> succ 9 + max 5 4 + 1
16
ghci> (max 5 4) + (succ 9) + 1
16
ghci> (succ 9) + (max 5 4) + 1
16
ghci> succ 9 * 10
100
ghci> succ (9 * 10)
91
ghci> div 92 10
9
ghci> div 3 4
0
ghci> div 4 3
1
ghci> 4 / 3
1.3333333333333333

ghci> mod 7 5
2
ghci> mod 7 5
2
ghci> mod 3 1
0
ghci> mod 7 2
1
ghci> x = 45
ghci> print x
45
ghci> return True
True
ghci> return False
False
ghci> x <- return 35
ghci> print x
35
ghci> putStrLn "Hello"
Hello
ghci> y <- getLine
Ajaykumar Nadar
ghci> print y
"Ajaykumar Nadar"
ghci>
```

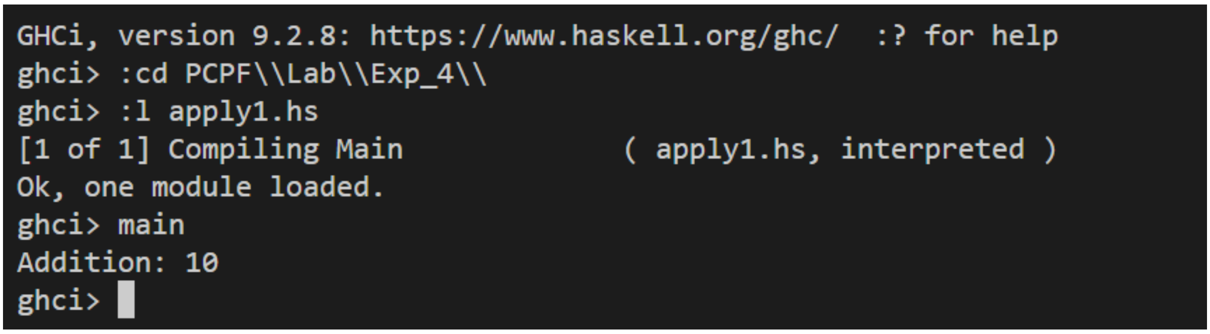
1. Write a function applyTwice to add and multiply two numbers that can take functions as parameters and also return functions.

Code:



```haskell
-- Author: Ajaykumar Nadar

applyTwice::(a->a) -> a -> a
applyTwice f x = f(f x)

main::IO()
main = do
  putStr "Addition: "
  print (applyTwice (+ 2) 6)
```

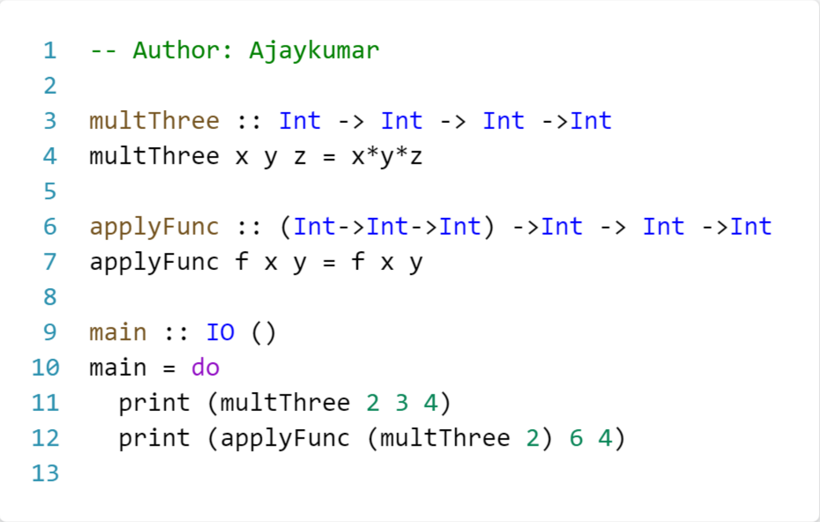
Output:



```
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :cd PCPF\\Lab\\Exp_4\\
ghci> :l apply1.hs
[1 of 1] Compiling Main             ( apply1.hs, interpreted )
Ok, one module loaded.
ghci> main
Addition: 10
ghci>
```

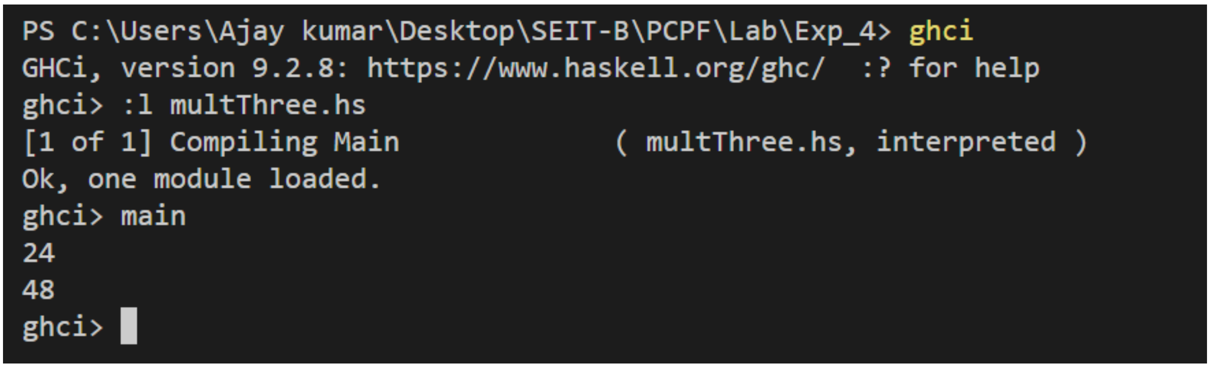
2. Write a function multThree ((multThree 3) 5) 9 that can take functions as parameters and also return functions.

Code:



```haskell
-- Author: Ajaykumar

multThree :: Int -> Int -> Int ->Int
multThree x y z = x*y*z

applyFunc :: (Int->Int->Int) ->Int -> Int ->Int
applyFunc f x y = f x y

main :: IO ()
main = do
  print (multThree 2 3 4)
  print (applyFunc (multThree 2) 6 4)

```

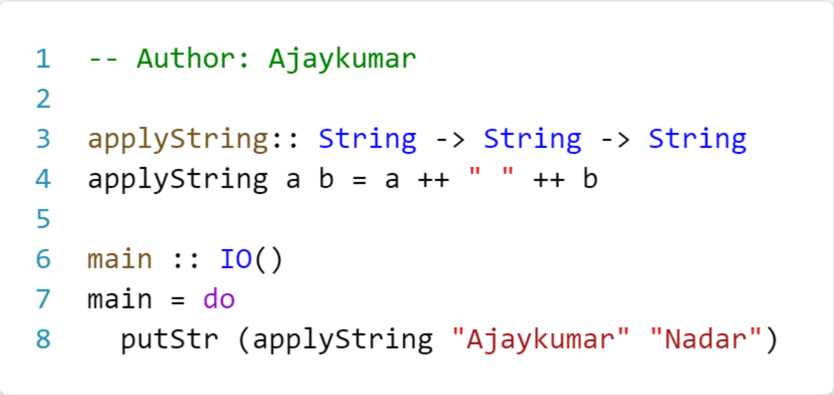
Output:



```
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 multThree.hs
[1 of 1] Compiling Main             ( multThree.hs, interpreted )
Ok, one module loaded.
ghci> main
24
48
ghci> 
```

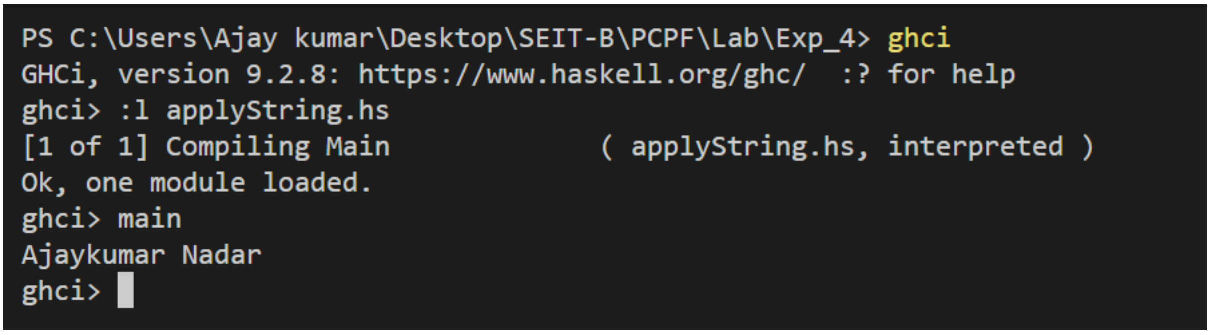
3. Write a function applyString to append two strings using higher order functions.

Code:



```haskell
-- Author: Ajaykumar

applyString:: String -> String -> String
applyString a b = a ++ " " ++ b

main :: IO()
main = do
  putStr (applyString "Ajaykumar" "Nadar")
```

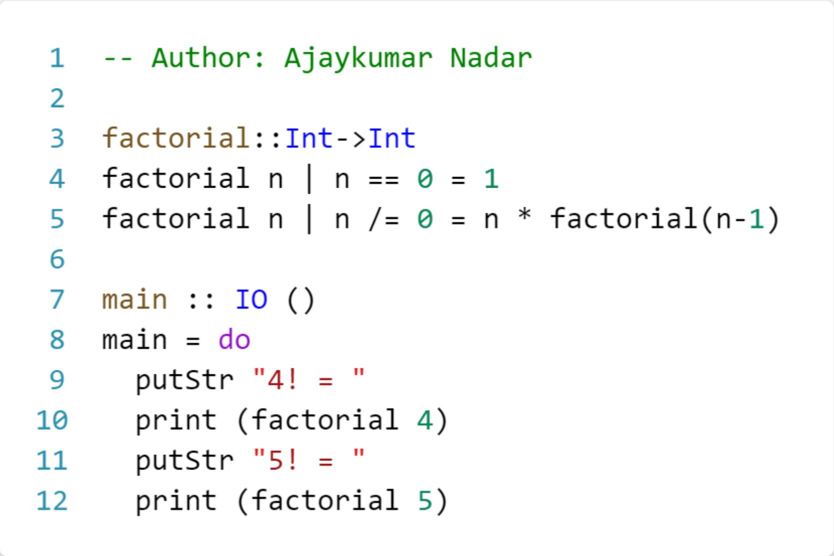
Output:



```
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 applyString.hs
[1 of 1] Compiling Main             ( applyString.hs, interpreted )
Ok, one module loaded.
ghci> main
Ajaykumar Nadar
ghci> 
```

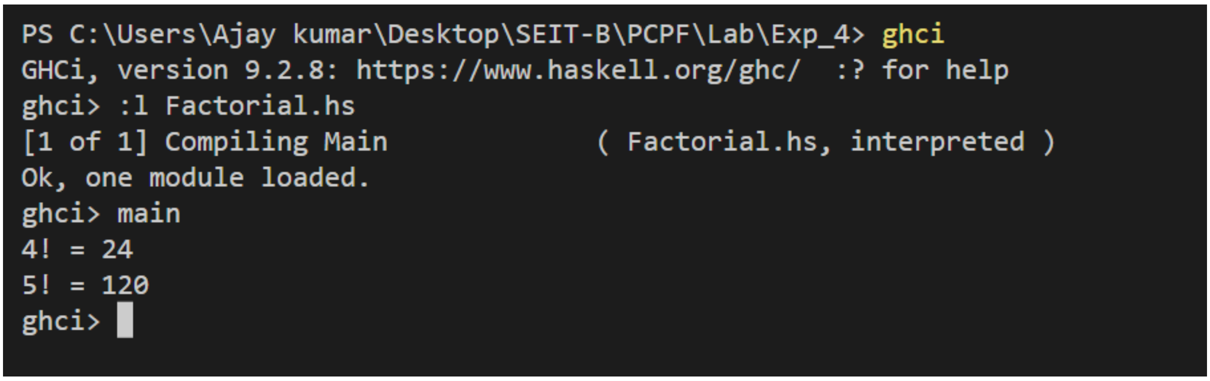
4. Write a program in Haskell to evaluate factorial of a number using recursion

Code:



```haskell
-- Author: Ajaykumar Nadar

factorial::Int->Int
factorial n | n == 0 = 1
factorial n | n /= 0 = n * factorial(n-1)

main :: IO ()
main = do
  putStr "4! = "
  print (factorial 4)
  putStr "5! = "
  print (factorial 5)
```

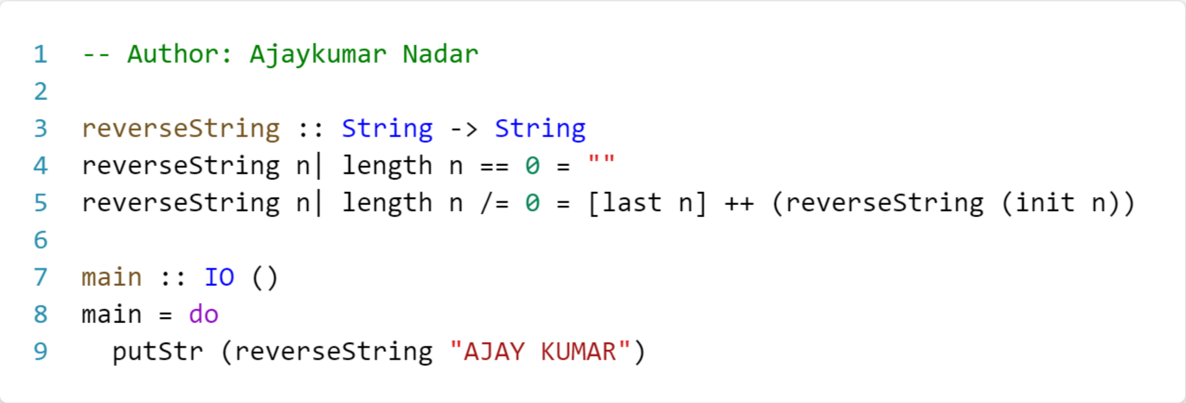
Output:



```
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 Factorial.hs
[1 of 1] Compiling Main             ( Factorial.hs, interpreted )
Ok, one module loaded.
ghci> main
4! = 24
5! = 120
ghci> 
```

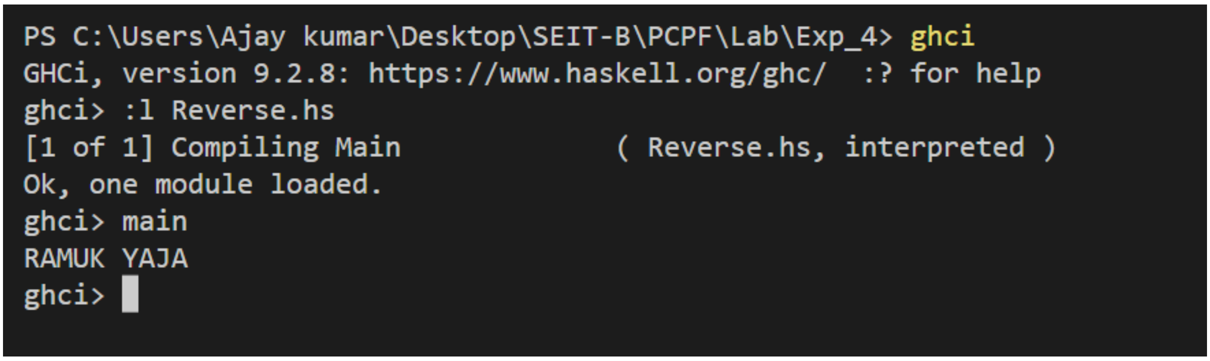
5. Write a program in Haskell to reverse a string using recursion

Code:



```haskell
-- Author: Ajaykumar Nadar

reverseString :: String -> String
reverseString n| length n == 0 = ""
reverseString n| length n /= 0 = [last n] ++ (reverseString (init n))

main :: IO ()
main = do
  putStr (reverseString "AJAY KUMAR")
```

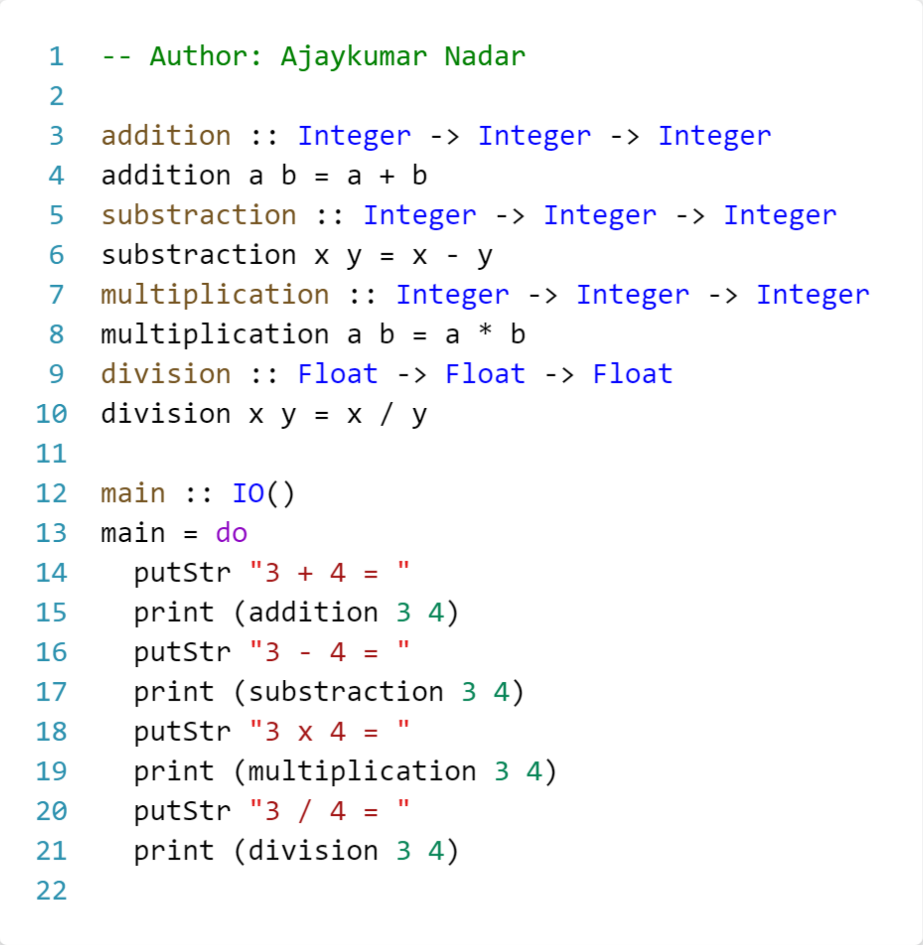
Output:



```
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 Reverse.hs
[1 of 1] Compiling Main             ( Reverse.hs, interpreted )
Ok, one module loaded.
ghci> main
RAMUK YAJA
ghci>
```

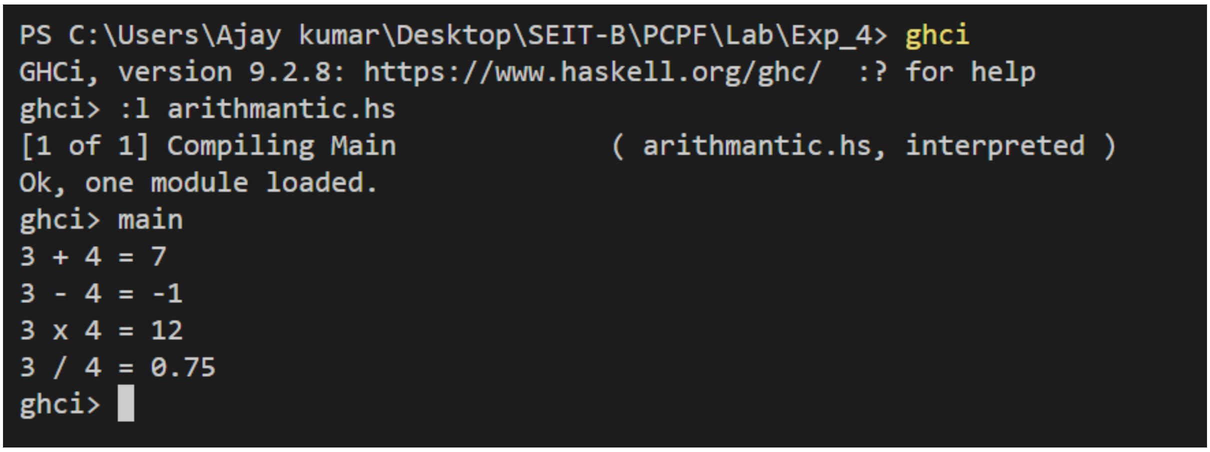
1. Addition, subtraction, multiplication and division of two numbers for number (integer) inputs

Code:



```haskell
-- Author: Ajaykumar Nadar

addition :: Integer -> Integer -> Integer
addition a b = a + b
substraction :: Integer -> Integer -> Integer
substraction x y = x - y
multiplication :: Integer -> Integer -> Integer
multiplication a b = a * b
division :: Float -> Float -> Float
division x y = x / y

main :: IO()
main = do
  putStr "3 + 4 = "
  print (addition 3 4)
  putStr "3 - 4 = "
  print (substraction 3 4)
  putStr "3 x 4 = "
  print (multiplication 3 4)
  putStr "3 / 4 = "
  print (division 3 4)

```

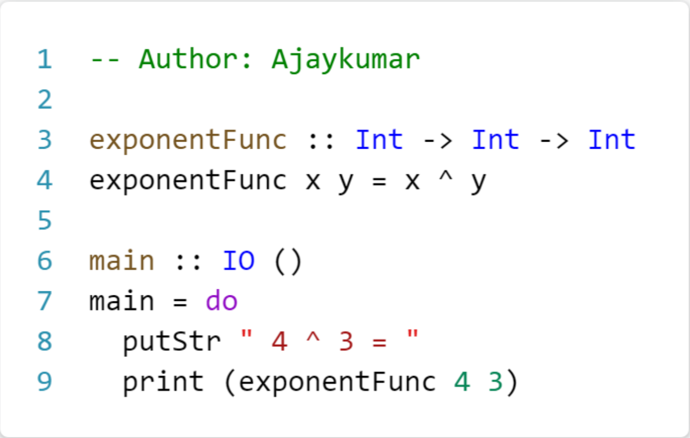
Output:



```
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 arithmantic.hs
[1 of 1] Compiling Main             ( arithmantic.hs, interpreted )
Ok, one module loaded.
ghci> main
3 + 4 = 7
3 - 4 = -1
3 x 4 = 12
3 / 4 = 0.75
ghci> █
```

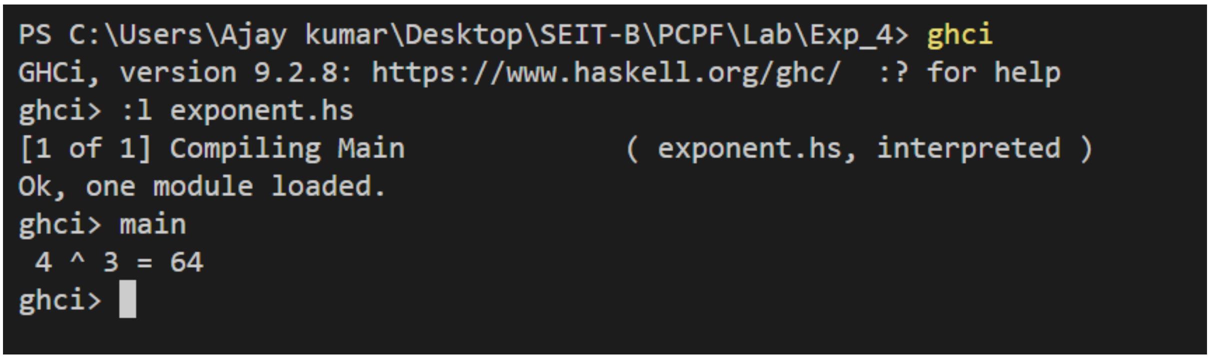
2. Exponent function x^y. The base and power are of type number (integer)

Code:



```haskell
-- Author: Ajaykumar

exponentFunc :: Int -> Int -> Int
exponentFunc x y = x ^ y

main :: IO ()
main = do
  putStr " 4 ^ 3 = "
  print (exponentFunc 4 3)
```

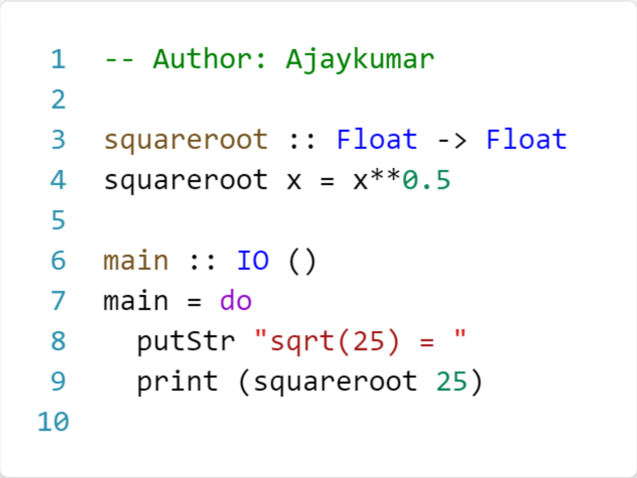
Output:



```
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 exponent.hs
[1 of 1] Compiling Main             ( exponent.hs, interpreted )
Ok, one module loaded.
ghci> main
 4 ^ 3 = 64
ghci> 
```

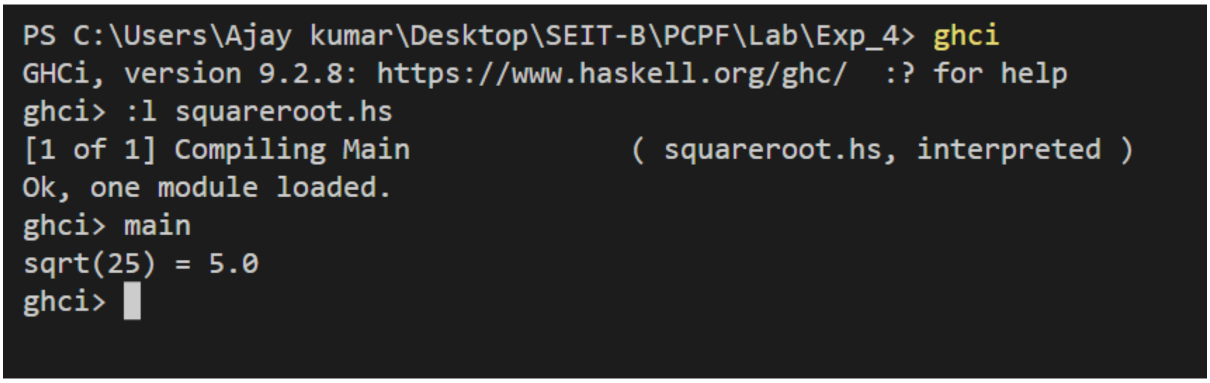
3. Square root of a number [x^0.5].

Code:



```haskell
-- Author: Ajaykumar

squareroot :: Float -> Float
squareroot x = x**0.5

main :: IO ()
main = do
  putStr "sqrt(25) = "
  print (squareroot 25)

```

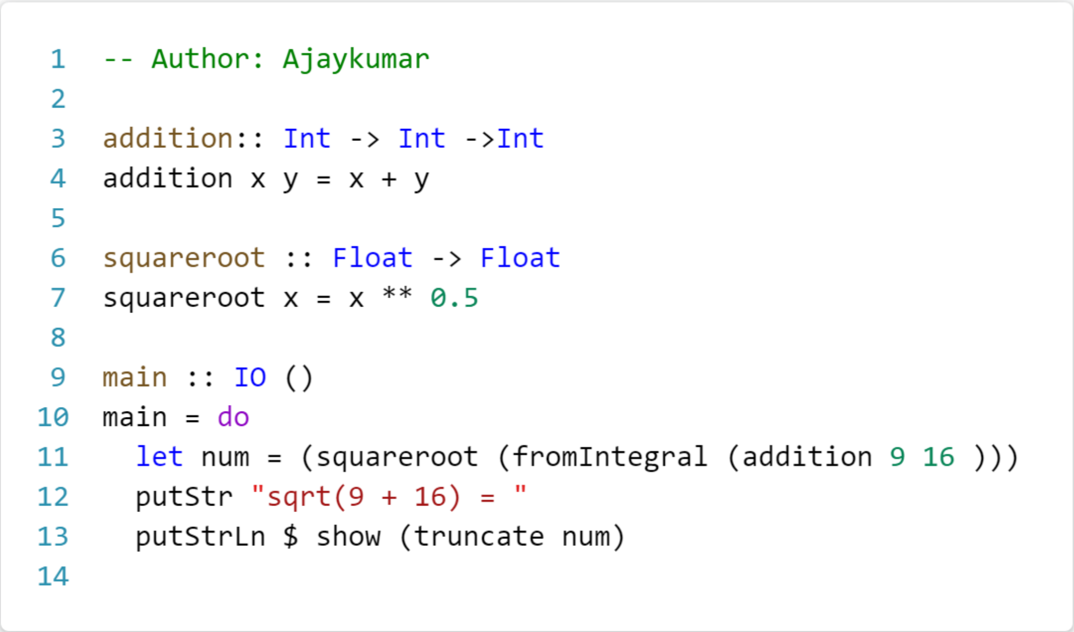
Output:



```
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 squareroot.hs
[1 of 1] Compiling Main             ( squareroot.hs, interpreted )
Ok, one module loaded.
ghci> main
sqrt(25) = 5.0
ghci> 
```

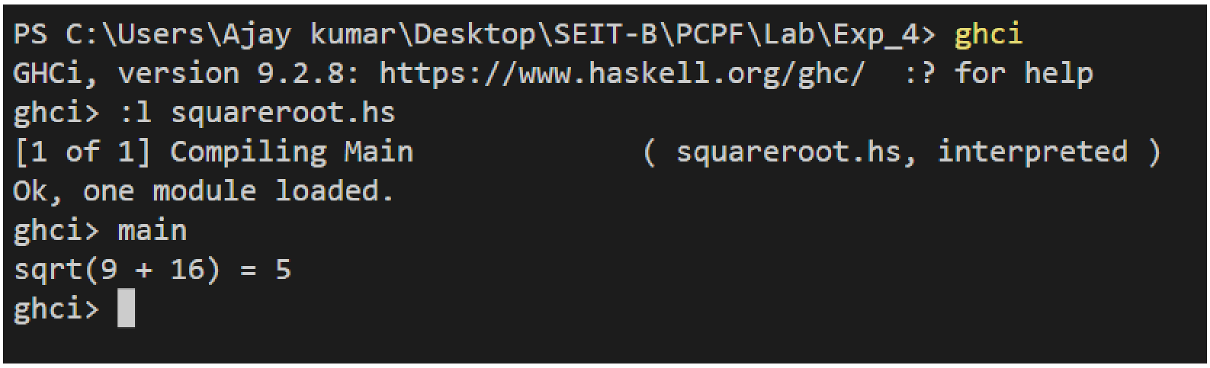
4. Add two numbers and then evaluate its square root.

Code:



```haskell
-- Author: Ajaykumar

addition:: Int -> Int ->Int
addition x y = x + y

squareroot :: Float -> Float
squareroot x = x ** 0.5

main :: IO ()
main = do
  let num = (squareroot (fromIntegral (addition 9 16 )))
  putStr "sqrt(9 + 16) = "
  putStrLn $ show (truncate num)

```

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 squareroot.hs
[1 of 1] Compiling Main             ( squareroot.hs, interpreted )
Ok, one module loaded.
ghci> main
sqrt(9 + 16) = 5
ghci> 
```