```
PS C:\Users\Ajay kumar\Desktop\SEIT-B> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help

ghci> x=[1,2,3]
ghci> print x
[1,2,3]

ghci> let x1 = ['a', 'b']
ghci> print x1
"ab"

ghci> [1,2,3,4]++[5,6,7]
[1,2,3,4,5,6,7]

ghci> [1.1, 2.1, 3.1, 4.1]++[5.1, 6.1, 7.1]
[1.1,2.1,3.1,4.1,5.1,6.1,7.1]

ghci> ['a', 'b'] ++ ['s', 'v']
"absv"

ghci> "hello" ++ " " ++ "world"
"hello world"

ghci> 'A': "Small Dog"
"ASmall Dog"

ghci> 'A':" " ++ "Small dog"
"A Small dog"

ghci> 'I':" "++"Small Dog"
"I Small Dog"

ghci> 6:[1,2,3,4]
[6,1,2,3,4]

ghci> 6.1: [1,2,3,4]
[6.1,1.0,2.0,3.0,4.0]

ghci> 6.1: [1,2,3,4] ++ 4:[1,2,3,4]
[6.1,1.0,2.0,3.0,4.0,4.0,1.0,2.0,3.0,4.0]

ghci> []
[]

ghci> "Steve Buscemi" !! 4
'e'
ghci> let x = "Steve Buscemi" !! 4
ghci> print x
```

```
'e'

ghci> let b = [[1,2,3], [4,5,6], [7,8,9]]
ghci> print b
[[1,2,3],[4,5,6],[7,8,9]]

ghci> b++[[1,1,1]]
[[1,2,3],[4,5,6],[7,8,9],[1,1,1]]

ghci> let y1 = b++[[1,1,1]]
ghci> print y1
[[1,2,3],[4,5,6],[7,8,9],[1,1,1]]

ghci> length [4,5,6,8]
4

ghci> length []
0

ghci> length [[]]
1

ghci> length [[[]]]
1

ghci> [[],[]]
[[],[]]

ghci> length [[],[]]
2

ghci> reverse [7, 5, 4, 9, 10]
[10,9,4,5,7]

ghci> take 3 [7,5,4,9,10]
[7,5,4]

ghci> drop 3 [7,5,4,9,10]
[9,10]

ghci> head [7,5,4,9,10]
7


ghci> tail [7,5,4,9,10]
[5,4,9,10]

ghci> init [7,5,4,9,10]
```

```
[7,5,4,9]

ghci> last [7,5,4,9,10]
10

ghci> sum [7,5,4,9,10]
35

ghci> product [7,5,4,9,10]
12600

ghci> maximum [4,6,7,8,9]
9

ghci> minimum[2,4,5,6,7]
2

ghci> 4 `elem` [6,7,8,4]
True

ghci> zipWith(+)[1,2,3][4,5,6]
[5,7,9]

ghci> zipWith(-)[1,2,3][4,5,6]
[-3,-3,-3]

ghci> zipWith(*)[1,2,3][4,5,6]
[4,10,18]

ghci> zipWith(/)[1,2,3][4,5,6]
[0.25,0.4,0.5]



ghci> [x*2 | x <- [1..10]]
[2,4,6,8,10,12,14,16,18,20]

ghci> [x*2 | x <- [1..100], x `mod` 2 == 0]
[4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,68,72,76,80,84,88,92,96,
100,104,108,112,116,120,124,128,132,136,140,144,148,152,156,160,164,168
,172,176,180,184,188,192,196,200]

ghci> [x | x<-[10..20], x /= 13, x /= 15, x /= 17]
[10,11,12,14,16,18,19,20]

ghci> [x^2 | x <- [1..10]]
[1,4,9,16,25,36,49,64,81,100]
```

```
ghci> [2^x | x <- [1..20]]
[2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768,65536,13107
2,262144,524288,1048576]

ghci> let nouns = ["hobo", "frog", "pope"]
ghci> let adj = ["lazy", "scram", "grouchy"]
ghci> [adj++" "++nouns|adj<-adj, nouns<-nouns]
["lazy hobo","lazy frog","lazy pope","scram hobo","scram frog","scram
pope","grouchy hobo","grouchy frog","grouchy pope"]

ghci> fst(8,9)
8

ghci> snd(8,0)
0

ghci> snd(8, "wow")
"wow"

ghci> zip [1,2,3,4][3,3,3,3]
[(1,3),(2,3),(3,3),(4,3)]

ghci> zip [1..5][3,3,3,3]
[(1,3),(2,3),(3,3),(4,3)]

ghci> zip [1..5]['a'..'e']
[(1,'a'),(2,'b'),(3,'c'),(4,'d'),(5,'e')]

ghci>
Leaving GHCi.
PS C:\Users\Ajay kumar\Desktop\SEIT-B>
```

1. Define a function doublePost that doubles the positive elements in the list of integers

**Code:**

```haskell
-- Author: Ajaykumar Nadar

doublePost :: [Int] -> [Int]
doublePost x = [2*y | y <- x, y>0]

main :: IO()
main = do
  let array = [-10, -7..15]
  let posArray = doublePost array
  print posArray
```

**Output:**

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_5> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l d
divisible.hs   doublePost.hs
ghci> :l doublePost.hs
[1 of 1] Compiling Main             ( doublePost.hs, interpreted )
Ok, one module loaded.
ghci> main
[4,10,16,22,28]
ghci>
```

2. Define a function spaces n which returns a string of n spaces

**Code:**

```haskell
-- Author: Ajaykumar Nadar

space:: Int->String
space n  = [' ' | x <- [1..n]]

main = do
    let x = space 5
    print x
```

**Output:**

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_5> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l spaces.hs
[1 of 1] Compiling Main             ( spaces.hs, interpreted )
Ok, one module loaded.
ghci> main
"     "
ghci>
```

3. Generate a list of triples (x,y,z) such that x2+y2=z2.

**Code:**

```haskell
-- Author: Ajaykumar Nadar

triples:: (Int, Int, Int) -> [(Int, Int, Int)]
triples (x, y, z) = [(m, n, o) | m <- [1..x], n <- [1..y],
                     o <- [1..z], ((m^2)+(n^2)) == (o^2), m <n, n<o]

main = do
  let x = triples (30, 40, 50)
  print x
```

**Output:**

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_5> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l triples.hs
[1 of 1] Compiling Main             ( triples.hs, interpreted )
Ok, one module loaded.
ghci> main
[(3,4,5),(5,12,13),(6,8,10),(7,24,25),(8,15,17),(9,12,15),(9,40,41)
,(10,24,26),(12,16,20),(12,35,37),(15,20,25),(15,36,39),(16,30,34),
(18,24,30),(20,21,29),(21,28,35),(24,32,40),(27,36,45),(30,40,50)]
ghci>
```

4. Define a function factor n which returns a list of integers that divide n. Omit the trival factors of 1 and n

**Code:**

```haskell
-- Author: Ajayumar Nadar

factor :: Int -> [Int]
factor n = [ x | x <- [2..(n-1)], n `mod` x == 0]

main :: IO()
main = do
  let x = factor 24
  print x
```

**Output:**

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_5> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l divisible.hs
[1 of 1] Compiling Main             ( divisible.hs, interpreted )
Ok, one module loaded.
ghci> main
[2,3,4,6,8,12]
ghci>
```

2. Write a Haskell snippet to create a list of first 10 numbers in numbers

**Code:**

```haskell
-- Author: Ajaykumar Nadar

main :: IO()
main = print [1..10]
```

**Output:**

3. Write a Haskell snippet to filter the elements from the list whose square root is greater than seven

**Code:**

```haskell
squareRoot :: [Int] -> [Int]
squareRoot ys = [ x | x <- ys, sqrt (fromIntegral x) > 7]

main :: IO()
main = do
  let array = squareRoot [43, 13, 64, 85, 38, 100, 81, 4, 1]
  print array
```

**Output:**

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_5> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l sqrt.hs
[1 of 1] Compiling Main             ( sqrt.hs, interpreted )
Ok, one module loaded.
ghci> main
[64,85,100,81]
ghci>
```

4. Write a Haskell snippet to implement Fibonacci series. Define an expression fibs :: [Integer] that generates this infinite sequence.

**Code:**

```haskell
-- Author: Ajaykumar Nadar

fibs :: [Integer]
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)

main :: IO ()
main = do
    print (take 10 fibs)

```

**Output:**

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\PCPF\Lab\Exp_5> ghci
GHCi, version 9.2.8: https://www.haskell.org/ghc/  :? for help
ghci> :l fibonacci.hs
[1 of 1] Compiling Main             ( fibonacci.hs, interpreted )
Ok, one module loaded.
ghci> main
[0,1,1,2,3,5,8,13,21,34]
ghci>
```