Practical 12: Haskell

ARITHMETIC OPERATIONS:

Prelude> 2022 - 2004

18

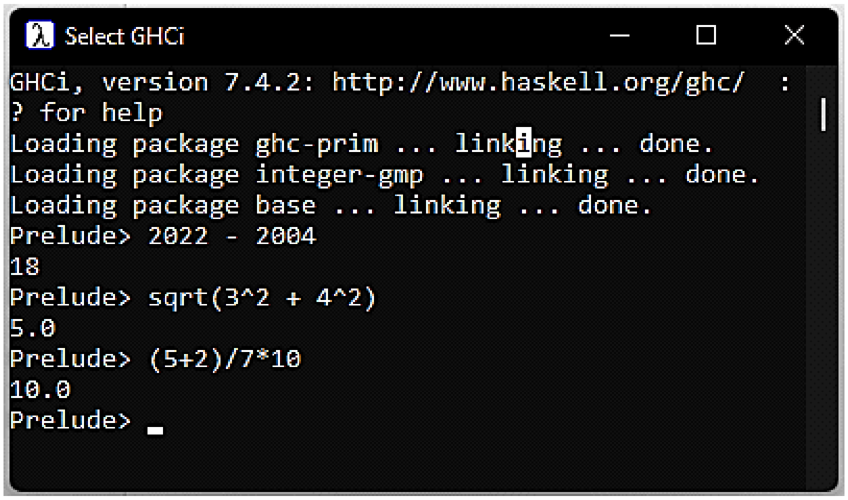
Prelude> sqrt(3^2 + 4^2)

5.0

Prelude> (5+2)/7*10

10.0

Snapshot:



```
GHCi, version 7.4.2: http://www.haskell.org/ghc/  :
? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> 2022 - 2004
18
Prelude> sqrt(3^2 + 4^2)
5.0
Prelude> (5+2)/7*10
10.0
Prelude>
```

COMPARISON OPERATIONS:

Prelude> "ABC" == "abc"

False

Prelude> 10>=5*2

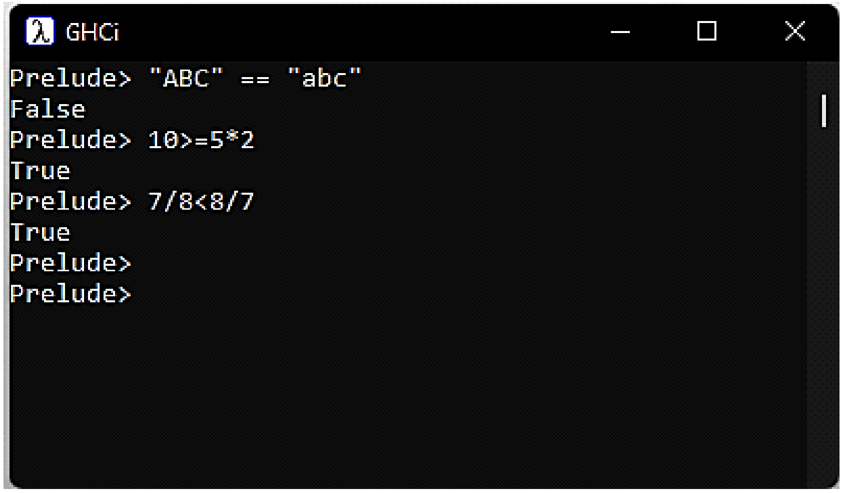
True

Prelude> 7/8<8/7

True

Prelude>

Snapshot:



```
Prelude> "ABC" == "abc"
False
Prelude> 10>=5*2
True
Prelude> 7/8<8/7
True
Prelude>
Prelude>
```

LIST OPERATIONS:

Prelude> [0,5..50]

[0,5,10,15,20,25,30,35,40,45,50]

Prelude> product[1..5]

120

Prelude> sum[1..5]

15

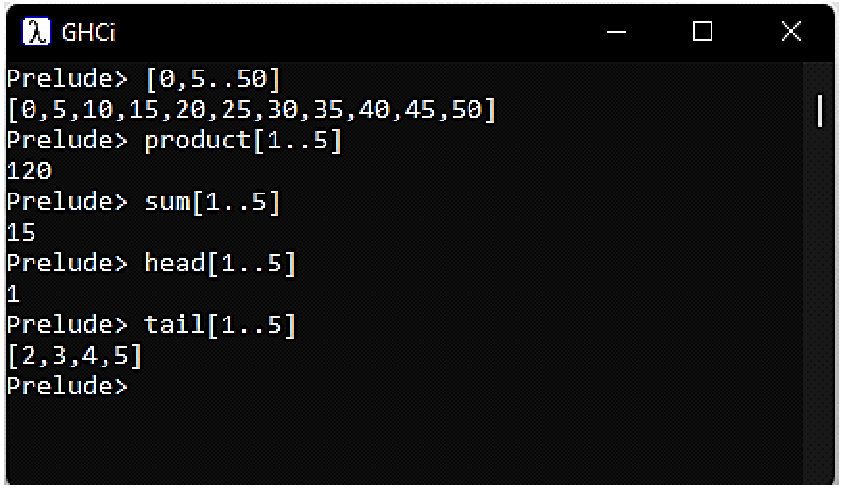
Prelude> head[1..5]

1

Prelude> tail[1..5]

[2,3,4,5]

Snapshot:



```
Prelude> [0,5..50]
[0,5,10,15,20,25,30,35,40,45,50]
Prelude> product[1..5]
120
Prelude> sum[1..5]
15
Prelude> head[1..5]
1
Prelude> tail[1..5]
[2,3,4,5]
Prelude>
```

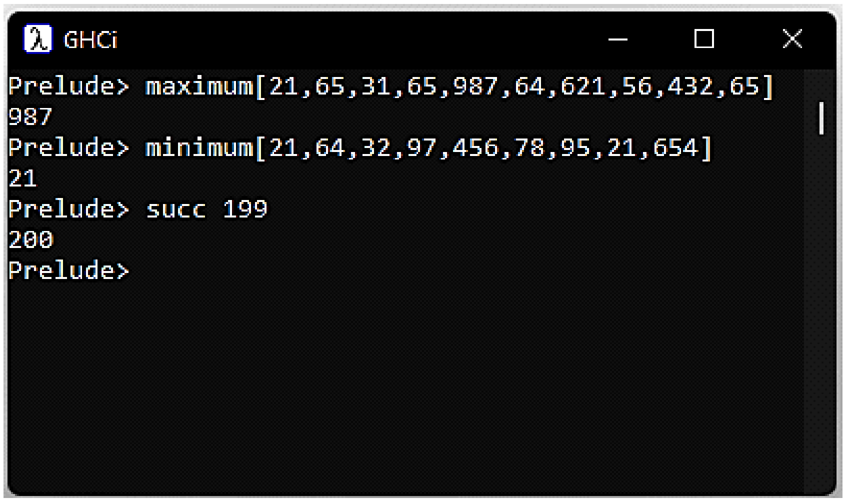
OTHER MATHETATICAL OPERATIONS:

Prelude> maximum[21,65,31,65,987,64,621,56,432,65]

987

```
Prelude> minimum[21,64,32,97,456,78,95,21,654]
21
Prelude> succ 199
200
```

Snapshot:

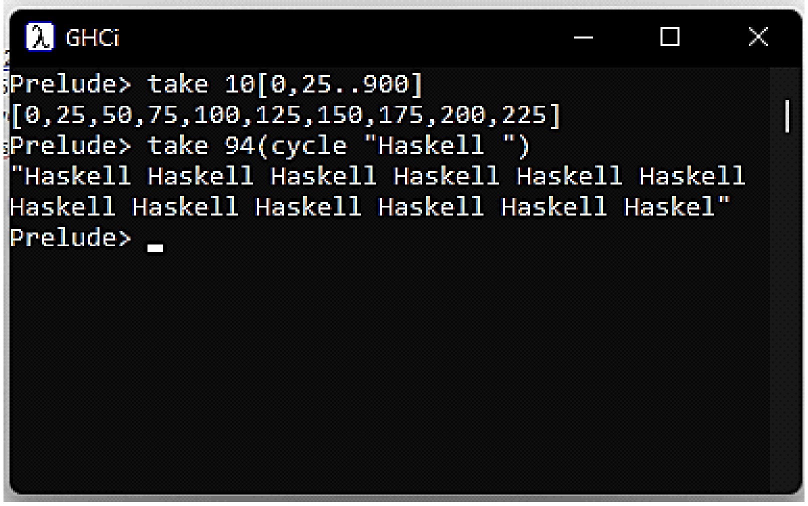


```
Prelude> maximum[21,65,31,65,987,64,621,56,432,65]
987
Prelude> minimum[21,64,32,97,456,78,95,21,654]
21
Prelude> succ 199
200
Prelude>
```

TAKE COMMAND:

```
Prelude> take 10[0,25..900]
[0,25,50,75,100,125,150,175,200,225]
Prelude> take 94(cycle "Haskell ")
"Haskell Haskell Haskell Haskell Haskell Haskell Haskell Haskell Haskell Haskell Haskell Haskel"
Prelude>
```

Snapshot:



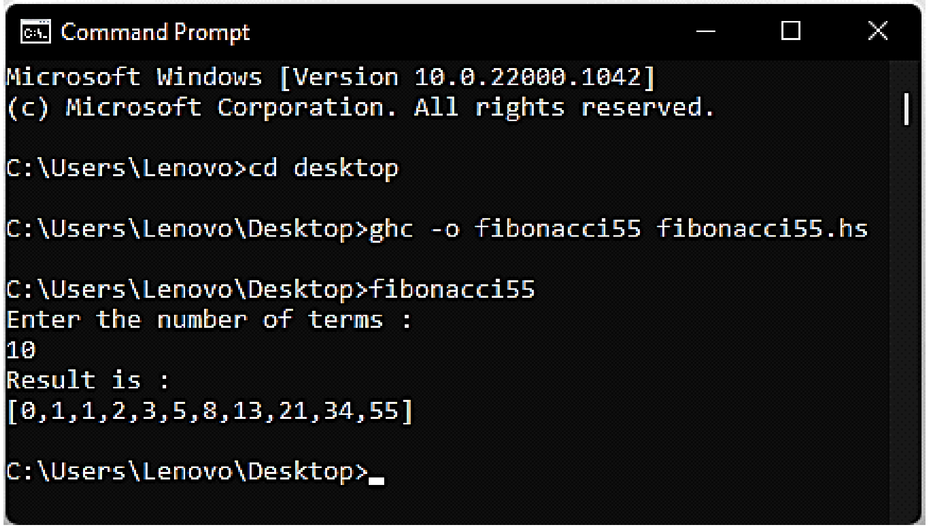
```
Prelude> take 10[0,25..900]
[0,25,50,75,100,125,150,175,200,225]
Prelude> take 94(cycle "Haskell ")
"Haskell Haskell Haskell Haskell Haskell Haskell
Haskell Haskell Haskell Haskell Haskell Haskel"
Prelude>
```

FIBONACCI SERIES:

Code:

```
Enter the number of terms :
10
Result is :
[0,1,1,2,3,5,8,13,21,34,55]
```

Snapshot:



```
Microsoft Windows [Version 10.0.22000.1042]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd desktop

C:\Users\Lenovo\Desktop>ghc -o fibonacci55 fibonacci55.hs

C:\Users\Lenovo\Desktop>fibonacci55
Enter the number of terms :
10
Result is :
[0,1,1,2,3,5,8,13,21,34,55]

C:\Users\Lenovo\Desktop>
```

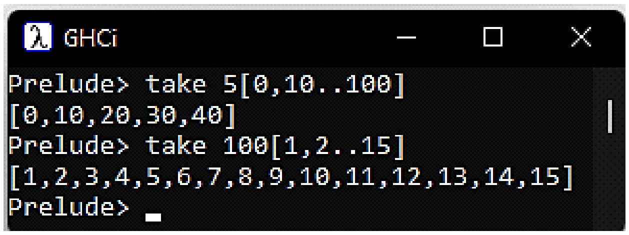
PRACTICAL- 13

a. Use of "TAKE" command.

Code:

```
Prelude> take 5[0,10..100]
[0,10,20,30,40]
Prelude> take 100[1,2..15]
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
```

Snapshot:



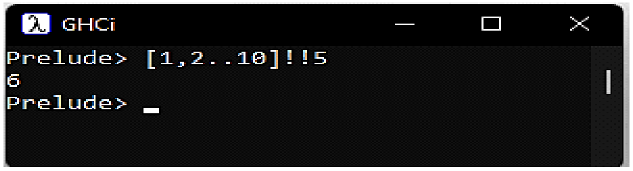
```
Prelude> take 5[0,10..100]
[0,10,20,30,40]
Prelude> take 100[1,2..15]
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
Prelude>
```

b. Use of a "!!" command

Code:

```
Prelude> [1,2..10]!!5
6
```

Snapshot:



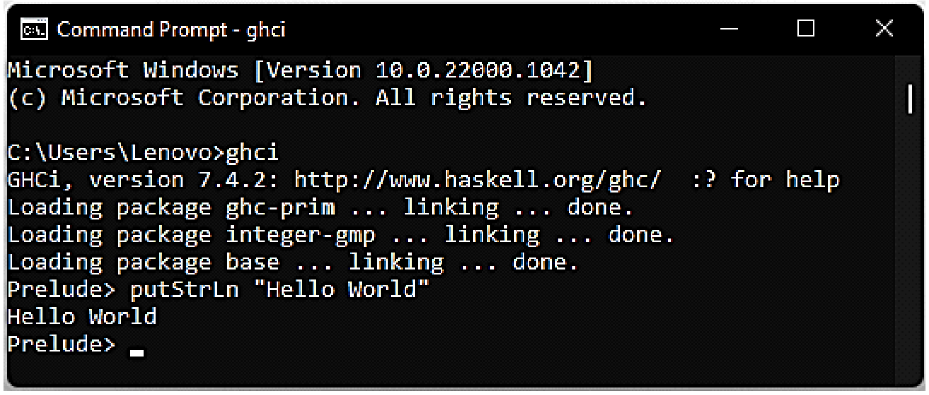
```
Prelude> [1,2..10]!!5
6
Prelude>
```

c. To print "Hello World" using cmd.

Code:

```
Prelude> putStrLn "Hello World"
Hello World
```

Snapshot:



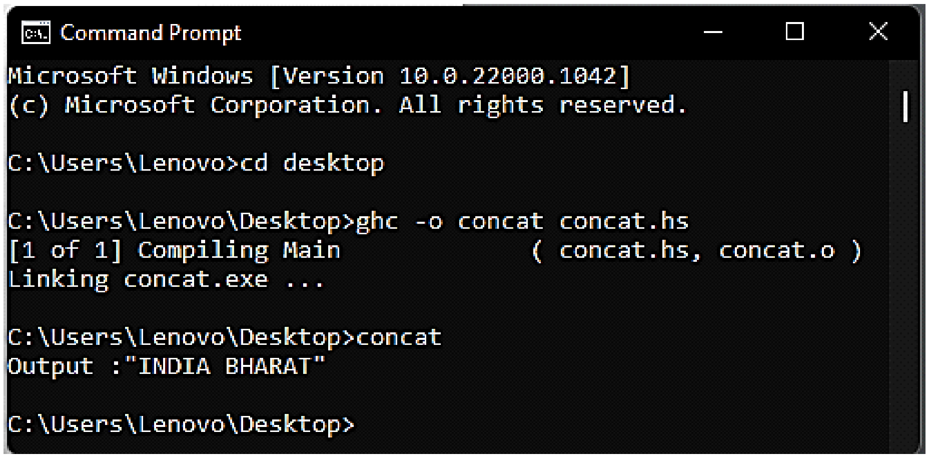
```
Microsoft Windows [Version 10.0.22000.1042]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>ghci
GHCi, version 7.4.2: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> putStrLn "Hello World"
Hello World
Prelude> _
```

d. To Concatenate two lists [‘I’,’N’, ‘D’, ‘I’, ‘A’] and [‘B’,‘H’, ‘A’, ‘R’ ,’A’,’T’] using Haskell

Code:

```
main = do
let list1 = "INDIA"
let list2 = "BHARAT"
putStr "Output :"
print(list1 ++ " " ++ list2)
```

Snapshot:



```
Microsoft Windows [Version 10.0.22000.1042]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd desktop

C:\Users\Lenovo\Desktop>ghc -o concat concat.hs
[1 of 1] Compiling Main             ( concat.hs, concat.o )
Linking concat.exe ...

C:\Users\Lenovo\Desktop>concat
Output :"INDIA BHARAT"

C:\Users\Lenovo\Desktop>
```

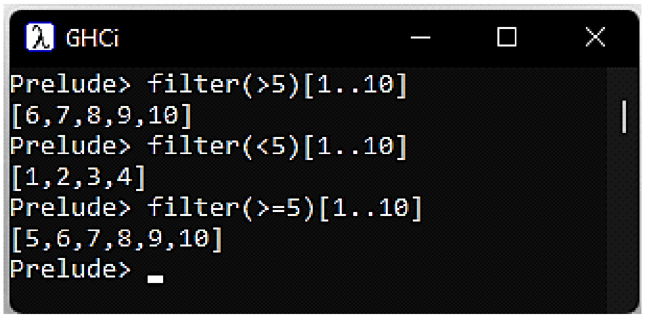
e. Use of filter ,map and other higher order functions

Filter:

Code:

```
Prelude> filter(>5)[1..10]
[6,7,8,9,10]
Prelude> filter(<5)[1..10]
[1,2,3,4]
Prelude> filter(>=5)[1..10]
[5,6,7,8,9,10]
```

Snapshot:



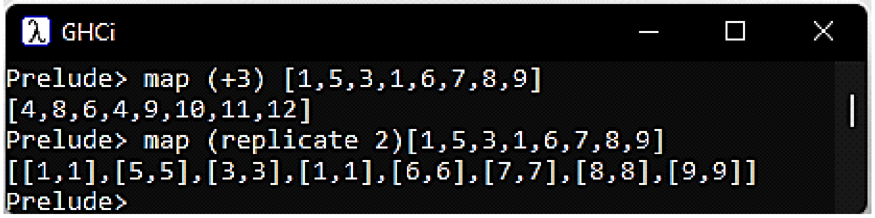
```
Prelude> filter(>5)[1..10]
[6,7,8,9,10]
Prelude> filter(<5)[1..10]
[1,2,3,4]
Prelude> filter(>=5)[1..10]
[5,6,7,8,9,10]
Prelude> _
```

Map:

Code:

```
Prelude> map (+3) [1,5,3,1,6,7,8,9]
[4,8,6,4,9,10,11,12]
Prelude> map (replicate 2)[1,5,3,1,6,7,8,9]
[[1,1],[5,5],[3,3],[1,1],[6,6],[7,7],[8,8],[9,9]]
```

Snapshot:



```
Prelude> map (+3) [1,5,3,1,6,7,8,9]
[4,8,6,4,9,10,11,12]
Prelude> map (replicate 2)[1,5,3,1,6,7,8,9]
[[1,1],[5,5],[3,3],[1,1],[6,6],[7,7],[8,8],[9,9]]
Prelude>
```

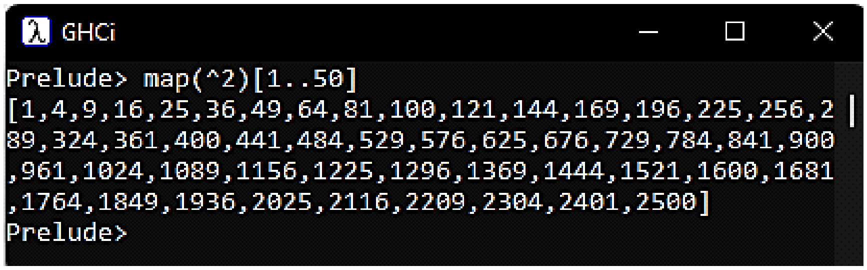
PRACTICAL- 14

a. Square of numbers from 1 to 50

Code:

```
Prelude> map(^2)[1..50]
[1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400,441,484,529,576,625,676,729,7
84,841,900,961,1024,1089,1156,1225,1296,1369,1444,1521,1600,1681,1764,1849,1936,2025,2116,220
9,2304,2401,2500]
```

Snapshot:

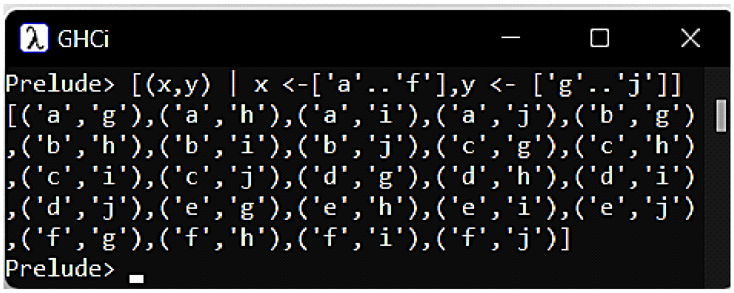


```
Prelude> map(^2)[1..50]
[1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,2
89,324,361,400,441,484,529,576,625,676,729,784,841,900
,961,1024,1089,1156,1225,1296,1369,1444,1521,1600,1681
,1764,1849,1936,2025,2116,2209,2304,2401,2500]
Prelude>
```

b. Generate the tuples out of [‘a’..’f’] and [‘g’..’j’]

Code: `[(x,y) | x <-['a'..'f'],y <- ['g'..'j']]`

Snapshot:

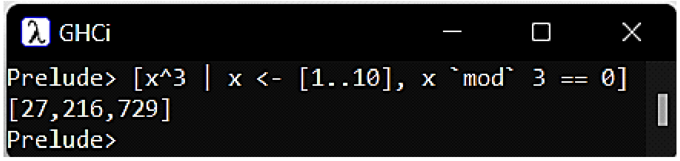


```
Prelude> [(x,y) | x <-['a'..'f'],y <- ['g'..'j']]
[('a','g'),('a','h'),('a','i'),('a','j'),('b','g')
,('b','h'),('b','i'),('b','j'),('c','g'),('c','h')
,('c','i'),('c','j'),('d','g'),('d','h'),('d','i')
,('d','j'),('e','g'),('e','h'),('e','i'),('e','j')
,('f','g'),('f','h'),('f','i'),('f','j')]
Prelude>
```

c. Generate the cube of number for every element from [1..10] such that cube mod 3 is zero

Code: ``[x^3 | x <- [1..10], x `mod` 3 == 0]``

Snapshot:



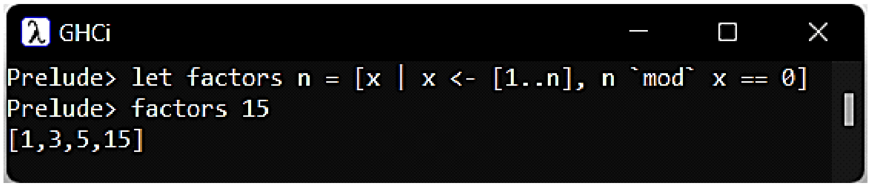
```
Prelude> [x^3 | x <- [1..10], x `mod` 3 == 0]
[27,216,729]
Prelude>
```

d. Write a function using list comprehension to generate factors of given number

Code: let factors n = [x | x <- [1..n], n `mod` x == 0]

factors 15

Snapshot:



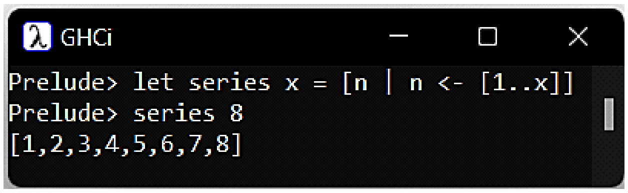
```
Prelude> let factors n = [x | x <- [1..n], n `mod` x == 0]
Prelude> factors 15
[1,3,5,15]
```

e. Write a function upto x that returns a list with all numbers between 1 and x

Code: let series x = [n | n <- [1..x]]

series 8

Snapshot:



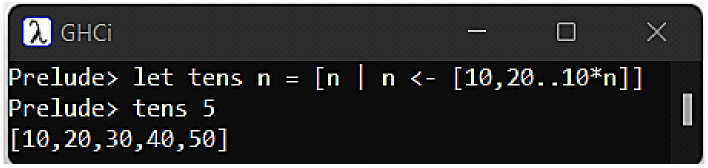
```
Prelude> let series x = [n | n <- [1..x]]
Prelude> series 8
[1,2,3,4,5,6,7,8]
```

f. Write a function tens n that returns a list containing [10, 20, ...] up to 10 * n

Code: let tens n = [n | n <- [10,20..10*n]]

tens 5

Snapshot:



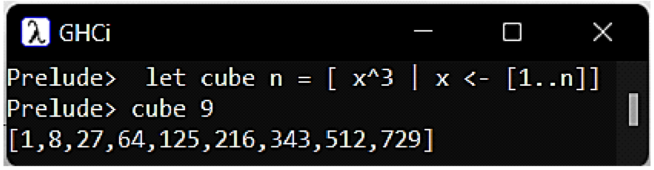
```
Prelude> let tens n = [n | n <- [10,20..10*n]]
Prelude> tens 5
[10,20,30,40,50]
```

g. Write a function cubes n that returns the list of all cubes up to n

Code: let cube n = [x^3 | x <- [1..n]]

cube 9

Snapshot:



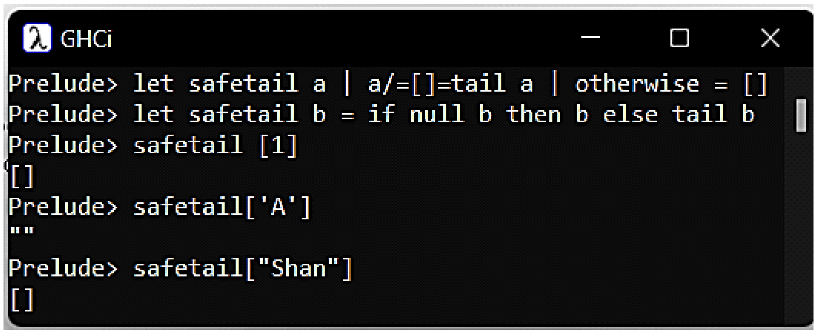
```
Prelude>  let cube n = [ x^3 | x <- [1..n]]
Prelude> cube 9
[1,8,27,64,125,216,343,512,729]
```

PRACTICAL- 15

Code:

```
Prelude> let safetail a | a/=[]=tail a | otherwise = []
Prelude> let safetail b = if null b then b else tail b
Prelude> safetail [1]
Prelude> safetail['A']
Prelude> safetail["Shan"]
```

Snapshot:



```
Prelude> let safetail a | a/=[]=tail a | otherwise = []
Prelude> let safetail b = if null b then b else tail b
Prelude> safetail [1]
[]
Prelude> safetail['A']
""
Prelude> safetail["Shan"]
[]
```