

Functional Programming Concept with **Haskell**

Tutorial for
Programming Languages Laboratory (CS 431)

July – November 2017

Dr. Samit Bhattacharya
Indian Institute of Technology Guwahati





Prelude

- Programming paradigms – broadly two types
 - Imperative
 - Declarative
- Imperative
 - Uses statements to change system state
 - Main concern – HOW to do things
- Declarative
 - Main concern – WHAT to do, not HOW to do things



Prelude

- Declarative paradigm
 - Logic programming – we have already seen it ([Prolog](#)); centered on RELATIONS
 - Functional programming – going to discuss in this tutorial



Functional Programming

- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism
- Characterized by the absence of “side effects”
 - Doesn’t rely on data outside the current function, and doesn’t change data that exists outside the current function

Non-functional

```
a = 0
def increment():
    global a
    a += 1
```

Functional

```
def increment(a):
    return a + 1
```



Functional Programming

➤ Languages that follow functional programming paradigm

- Haskell
- LISP
- Python
- Erlang
- Racket
- F#
- Clojure



Functional Programming

➤ Languages that follow functional programming paradigm

- Haskell
- LISP
- Python
- Erlang
- Racket
- F#
- Clojure

we are going with Haskell this time



Haskell

- Standardized purely functional programming language
- Named after logician and mathematician Haskell Brooks Curry
- History
 - First version (“Haskell 1.0”) was introduced in 1990
 - The latest standard of Haskell is “Haskell 2010”
- Applications example
 - Darcs is a revision control system written in Haskell, with several innovative features, such as more precise control of the patches to be applied
 - Xmonad is a window manager for the X Window System, written fully in Haskell
 - Facebook implements its anti-spam programs in Haskell, as open-source software



Haskell - Features

- Statically typed
- Purely functional
- Type inference
- Lazy
- Concurrent
- Packages



Haskell - Features

- Statically typed
- Every expression in Haskell has a type which is determined at compile time
- The compiler knows which piece of code is a number, which is a string and so on, a lot of possible errors are caught at compile time
- All the types composed together by function application have to match up. If they don't, the program will be rejected by the compiler
- Types become not only a form of guarantee, but a language for expressing the construction of programs



Haskell - Features

- Purely functional
- Every function in Haskell is a function in the mathematical sense (i.e., "pure")
- Even side-effecting IO operations are but a description of what to do, produced by pure code
- There are no statements or instructions, only expressions which cannot mutate variables (local or global) nor access state like time or random numbers



Haskell - Features

- Type inference
 - You don't have to explicitly label every piece of code with a type because the type system can intelligently figure out a lot about it; if you say `a = 5 + 4`, you don't have to tell Haskell that `a` is a number
 - However, you can write out types if you choose, or ask the compiler to write them for you for handy documentation



Haskell - Features

- Lazy
 - Unless specifically told otherwise, Haskell won't execute functions and calculate things until it's really forced to show you a result
 - That goes well with referential transparency and it allows you to think of programs as a series of *transformations on data*



Haskell - Features

➤ Concurrent

- Haskell lends itself well to concurrent programming due to its explicit handling of effects
- Its flagship compiler, GHC, comes with a high-performance parallel garbage collector and light-weight concurrency library containing a number of useful concurrency primitives and abstractions



Haskell - Features

➤ Packages

- Open source contribution to Haskell is very active with a wide range of packages available on the public package servers
- There are 6,954 packages freely available; for instances

<u>bytestring</u>	Binary data	<u>base</u>	Prelude, IO, threads
<u>network</u>	Networking	<u>text</u>	Unicode text
<u>parsec</u>	Parser library	<u>directory</u>	File/directory
<u>hspec</u>	RSpec-like tests	<u>attoparsec</u>	Fast parser
<u>monad-logger</u>	Logging	<u>persistent</u>	Database ORM
<u>template-haskell</u>	Meta-programming	<u>tar</u>	Tar archives

Lets Start

Lets try to understand basic features of Haskell with examples



Run your First Haskell Program

➤ Download and Install Haskell

- Download link <https://www.haskell.org/downloads>

➤ File extension .hs

- Open text editor, write your program, save your program with .hs extension (e.g., haskell-tutorial.hs)

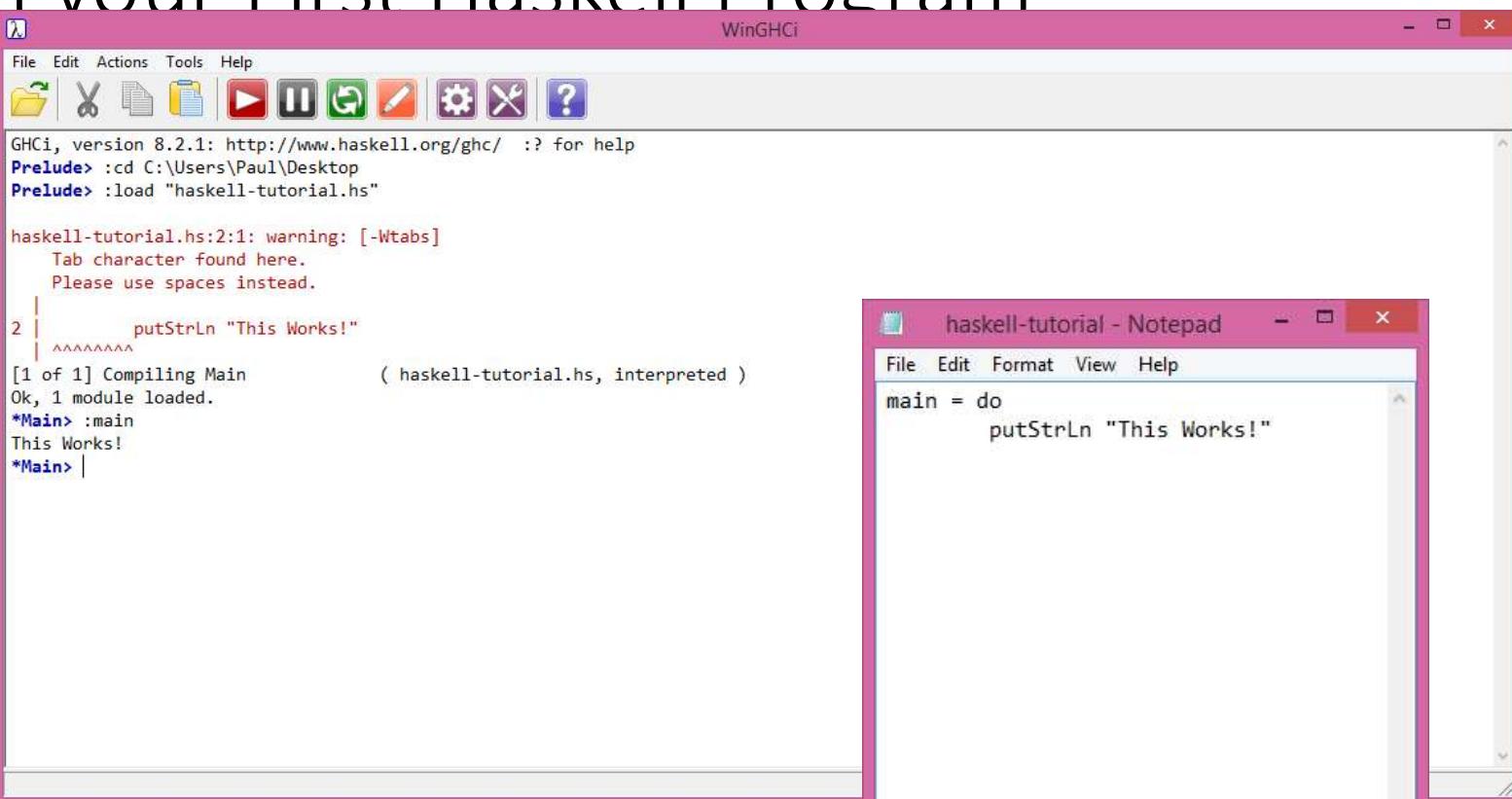
➤ Compilation and Run

➤ For Windows OS

- Open WinGHCi from start menu
- Load your program (File -> Load..)
- Run the function you want

Run your First Haskell Program

► Do



```

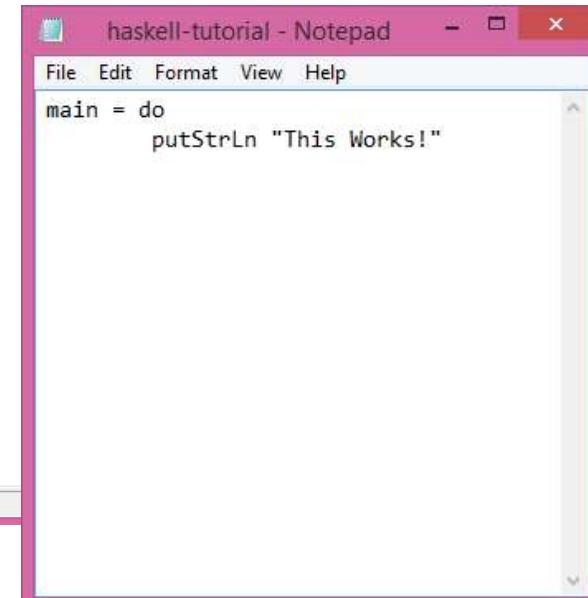
WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/  :? for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"

haskell-tutorial.hs:2:1: warning: [-Wtabs]
  Tab character found here.
  Please use spaces instead.

2 |     putStrLn "This Works!"
 | ^^^^^^^^^^
[1 of 1] Compiling Main           ( haskell-tutorial.hs, interpreted )
Ok, 1 module loaded.
*Main> :main
This Works!
*Main>

```

► File



```

haskell-tutorial - Notepad
File Edit Format View Help
main = do
    putStrLn "This Works!"

```

► Co

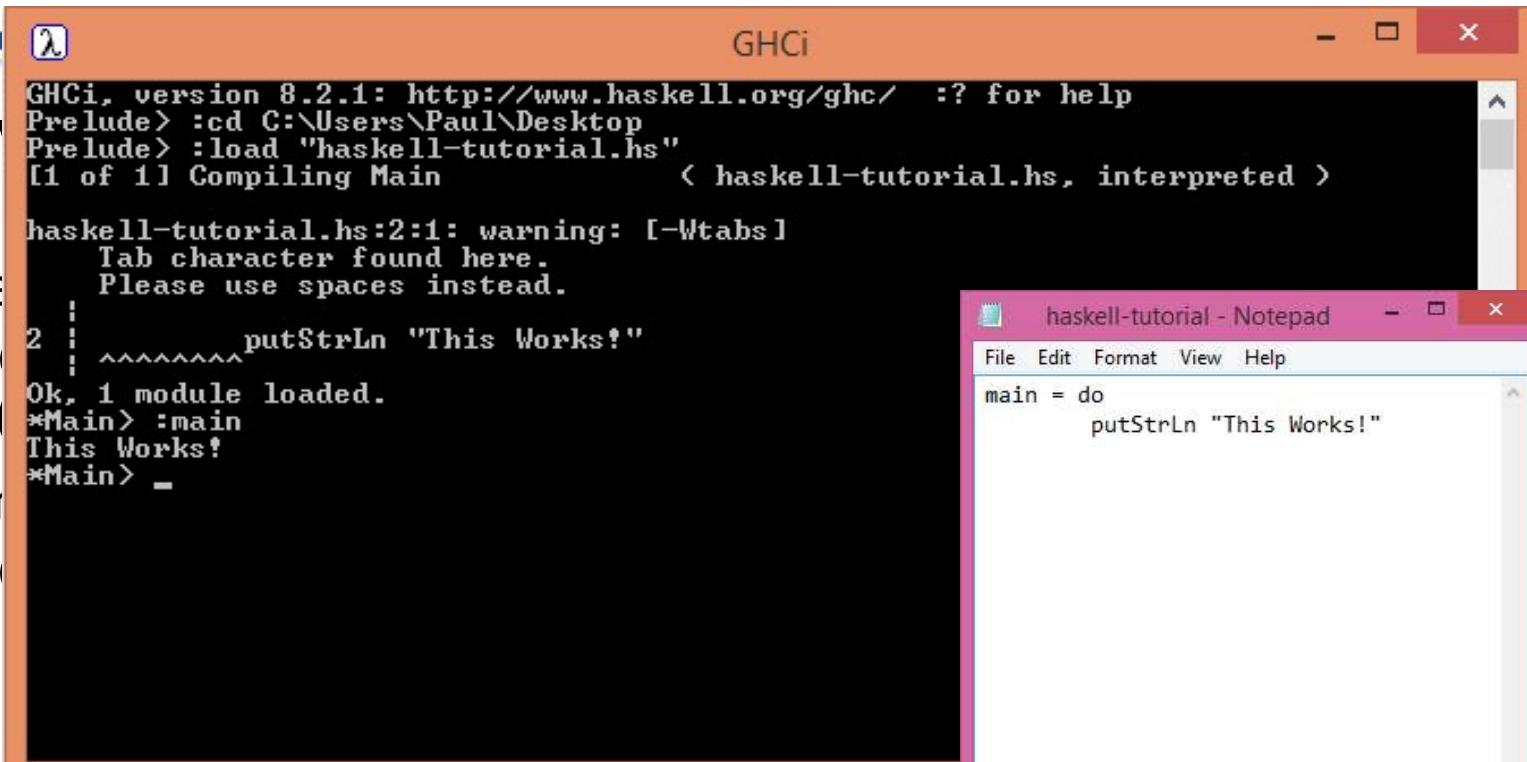


Run your First Haskell Program

- Download and Install Haskell
 - Download link <https://www.haskell.org/downloads>
- File extension .hs
 - Open text editor, write your program, save your program with .hs extension (e.g., haskell-tutorial.hs)
- Compilation and Run
 - Otherwise
 - Open GHCI
 - Enter into directory where you saved your program (`:cd C:\Users\Paul\Desktop`)
 - Load your program (`:load "haskell-tutorial.hs"`)
 - Run the function you want

Run your First Haskell Program

- Doing
- File
- Configuration
- Compiler
-



The image shows two windows side-by-side. On the left is a terminal window titled "GHCi" with a blue border. It displays the following Haskell code and its execution:

```
λ GHCi, version 8.2.1: http://www.haskell.org/ghc/  :? for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             < haskell-tutorial.hs, interpreted >
haskell-tutorial.hs:2:1: warning: [-Wtabs]
    Tab character found here.
    Please use spaces instead.
2 |     putStrLn "This Works!"
   | ^^^^^^^^^^
Ok, 1 module loaded.
*Main> :main
This Works!
*Main>
```

On the right is a "Notepad" window titled "haskell-tutorial - Notepad" with a pink border. It contains the same Haskell code:

```
File Edit Format View Help
main = do
    putStrLn "This Works!"
```

- Run the function you want



Few things you may keep in mind

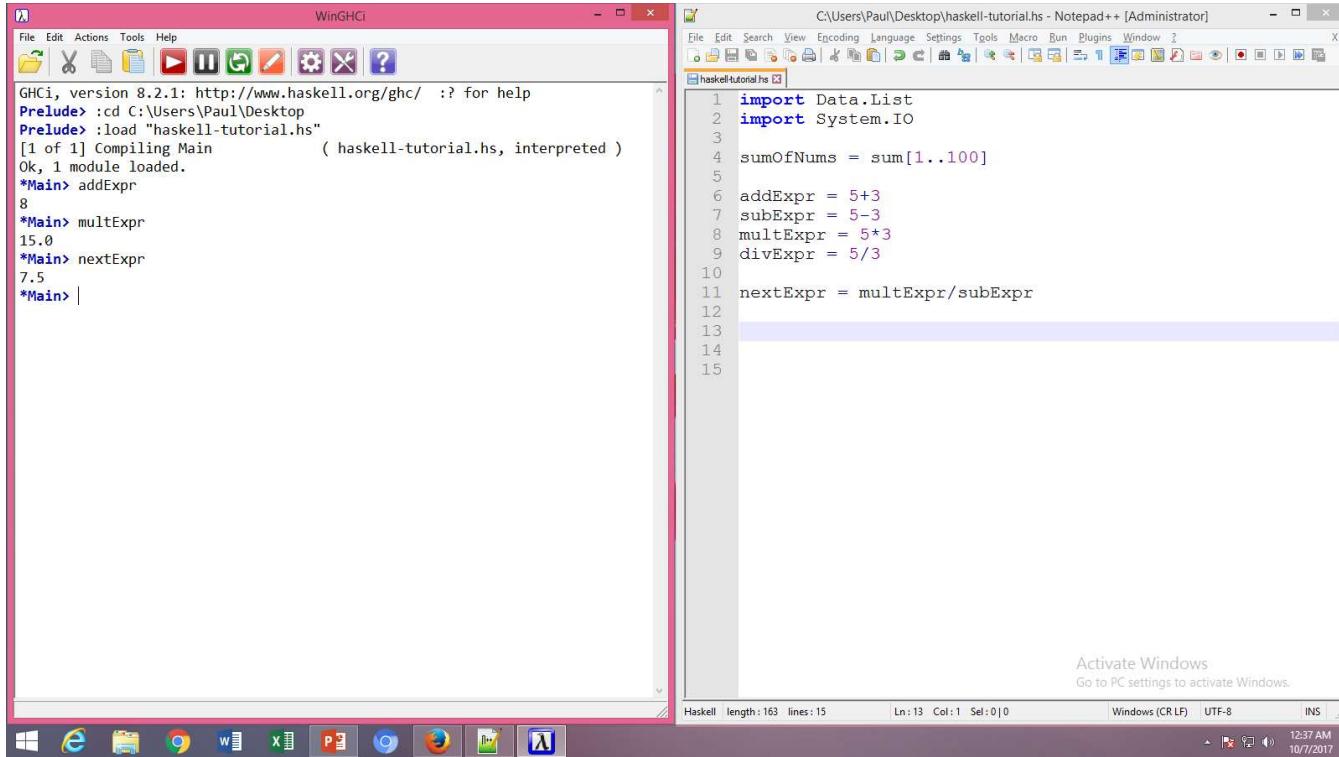
- Once you modify your program
 - Save it
 - Before running its function, recompile it - reload (*main> :r)
- Comment Line
 - --Comment
 - {-
 - Multiple Comments
 - }
- Clear Screen
 - Ctrl+S



Date Types

- Haskell uses type inference
 - Range of 'Int': -2^{63} to 2^{63}
 - Range of 'Integer': Unbound -- as per the capability of memory of the system
 - Other data types: Double, Bool, Char, Tuple -- will be discussing with example
 - always: always5 :: Int
 - always5 = 5 --Never Change

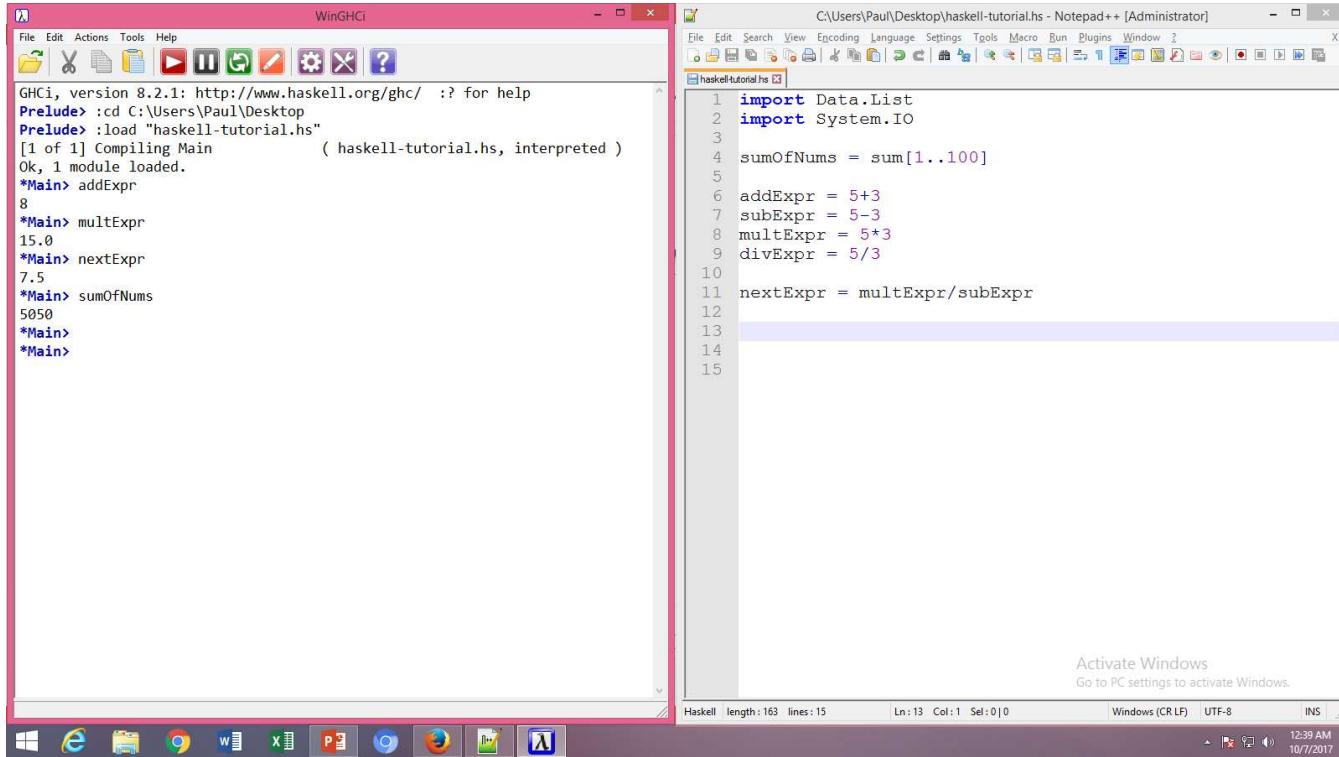
Expressions



The image shows a Windows desktop environment with two open windows:

- WinGHCi**: A terminal window showing the Haskell Interactive Component (GHCi) version 8.2.1. The session starts with the Prelude, changes the working directory to the user's desktop, loads a file named "haskell-tutorial.hs", and then enters a command loop where it defines and evaluates expressions like addExpr (5+3), subExpr (5-3), multExpr (5*3), and divExpr (5/3). It also defines nextExpr as multExpr divided by subExpr.
- Notepad++**: A code editor window titled "haskell-tutorial.hs" containing the source code for the Haskell program. The code imports Data.List and System.IO, defines sumOfNums as the sum of integers from 1 to 100, and defines four functions: addExpr, subExpr, multExpr, and divExpr. It then defines nextExpr as multExpr divided by subExpr.

Expressions



The image shows two windows side-by-side. On the left is the WinGHCi window, which displays a Haskell session. On the right is the Notepad++ window, which contains a Haskell script named `haskell-tutorial.hs`.

WinGHCi Window Content:

```

GHCi, version 8.2.1: http://www.haskell.org/ghc/ :? for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main           ( haskell-tutorial.hs, interpreted )
Ok, 1 module loaded.
*Main> addExpr
8
*Main> multExpr
15.0
*Main> nextExpr
7.5
*Main> sumOfNums
5050
*Main>
*Main>

```

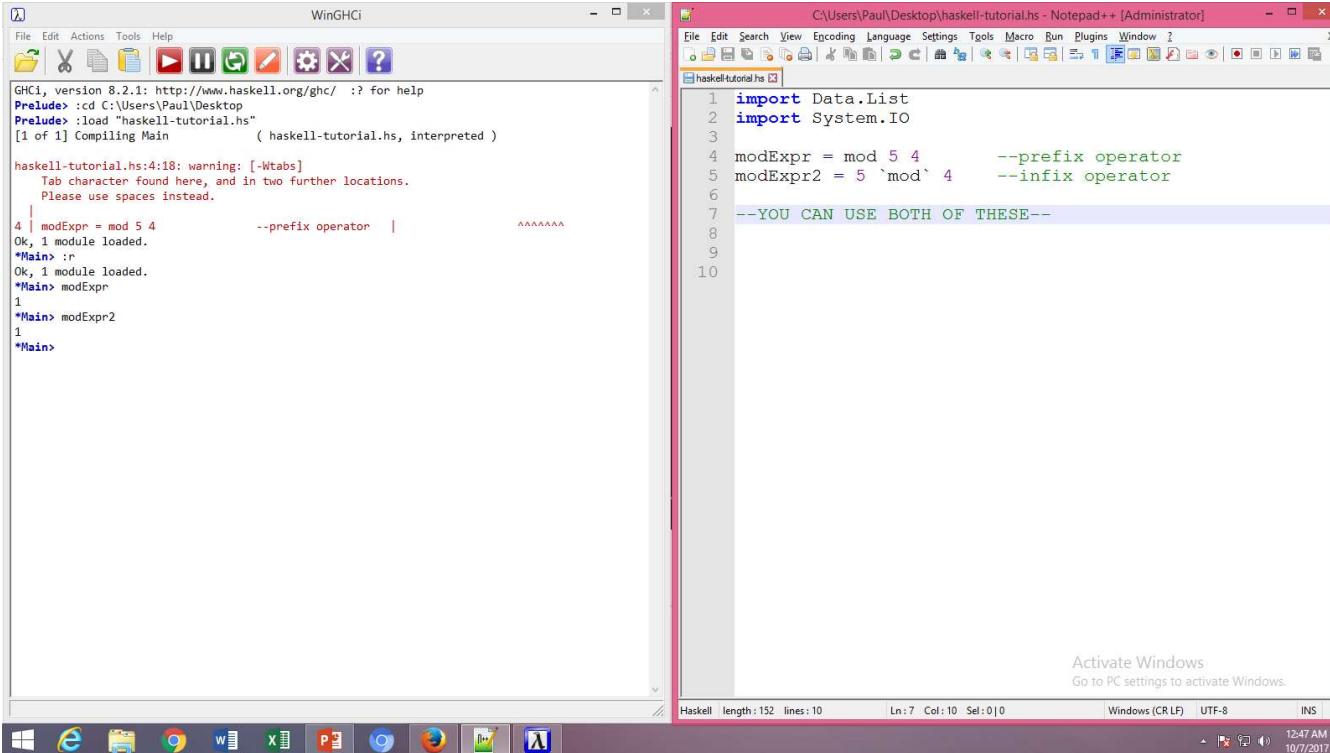
Notepad++ Window Content (`haskell-tutorial.hs`):

```

1 import Data.List
2 import System.IO
3
4 sumOfNums = sum[1..100]
5
6 addExpr = 5+3
7 subExpr = 5-3
8 multExpr = 5*3
9 divExpr = 5/3
10
11 nextExpr = multExpr/subExpr
12
13
14
15

```

Infix and Prefix Operator



The image shows two windows side-by-side. On the left is the WinGHCi terminal window, which displays Haskell code being run and its output. On the right is the Notepad++ editor window, which contains a Haskell script illustrating prefix and infix operators.

WinGHCi Terminal Output:

```

GHCi, version 8.2.1: http://www.haskell.org/ghc/ :? for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             ( haskell-tutorial.hs, interpreted )

haskell-tutorial.hs:4:18: warning: [-Wtabs]
    Tab character found here, and in two further locations.
    Please use spaces instead.

4 | modExpr = mod 5 4          --prefix operator  |
Ok, 1 module loaded.
*Main> :r
Ok, 1 module loaded.
*Main> modExpr
1
*Main> modExpr2
1
*Main>

```

Notepad++ Editor Content:

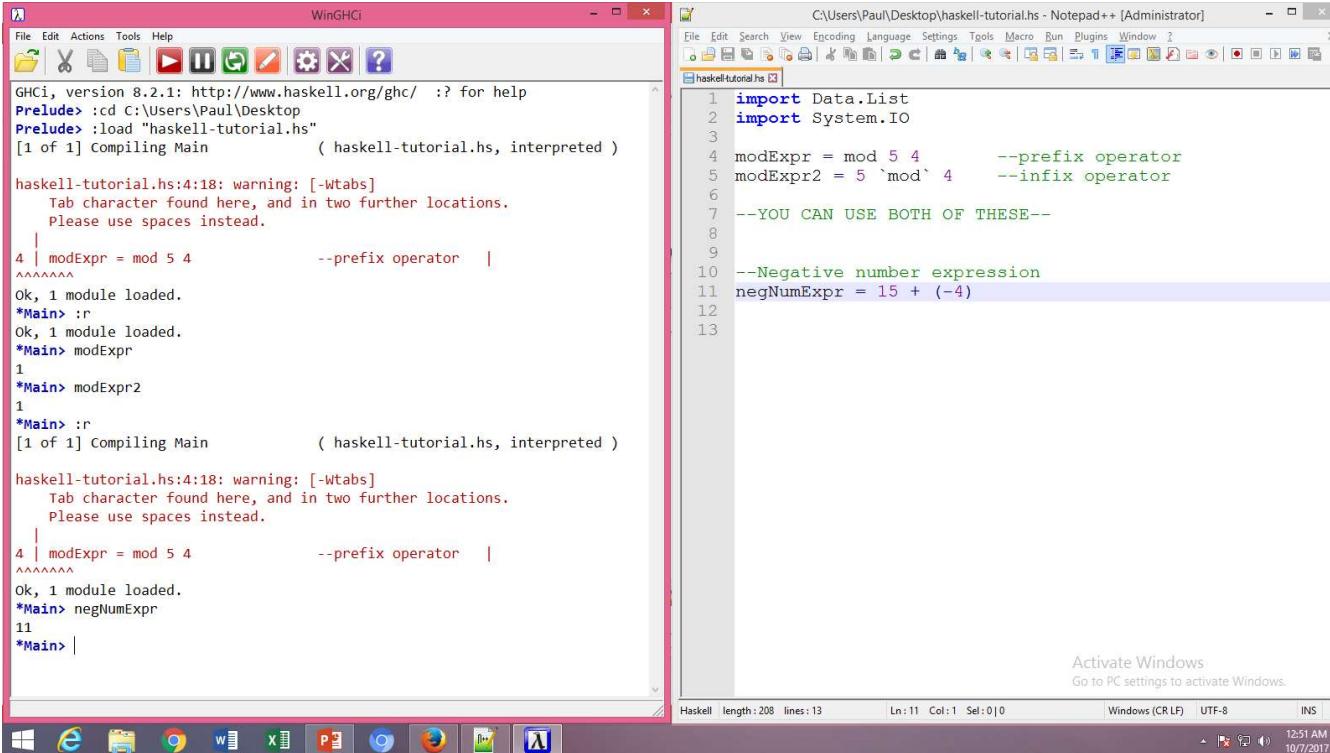
```

C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator] - x

1 import Data.List
2 import System.IO
3
4 modExpr = mod 5 4          --prefix operator
5 modExpr2 = 5 `mod` 4        --infix operator
6
7 --YOU CAN USE BOTH OF THESE--
8
9
10

```

Negative Number Expression



```

WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/ :? for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main           ( haskell-tutorial.hs, interpreted )

haskell-tutorial.hs:4:18: warning: [-Wtabs]
    Tab character found here, and in two further locations.
    Please use spaces instead.
  4 | modExpr = mod 5 4          --prefix operator |
~~~~~
Ok, 1 module loaded.
*Main> :r
Ok, 1 module loaded.
*Main> modExpr
1
*Main> modExpr2
1
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutorial.hs, interpreted )

haskell-tutorial.hs:4:18: warning: [-Wtabs]
    Tab character found here, and in two further locations.
    Please use spaces instead.
  4 | modExpr = mod 5 4          --prefix operator |
~~~~~
Ok, 1 module loaded.
*Main> negNumExpr
11
*Main> |

C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs [x]
1 import Data.List
2 import System.IO
3
4 modExpr = mod 5 4          --prefix operator
5 modExpr2 = 5 `mod` 4        --infix operator
6
7 --YOU CAN USE BOTH OF THESE--
8
9
10 --Negative number expression
11 negNumExpr = 15 + (-4)
12
13

Activate Windows
Go to PC settings to activate Windows.

Haskell length: 208 lines: 13 Ln:11 Col:1 Sel:0|0 Windows (CR LF) UTF-8 INS 12:51 AM 10/7/2017

```

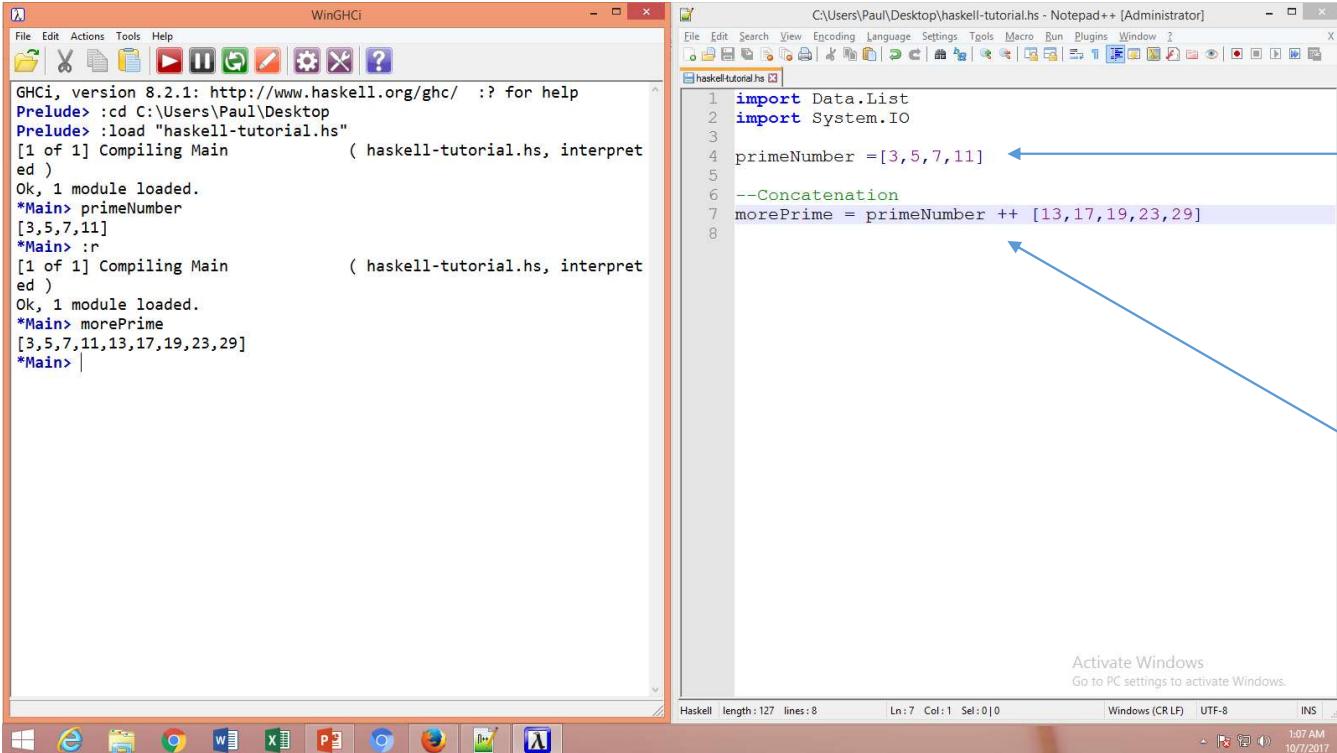


Other built-in Math Function

- `piVal = pi`
- `ePow9 = exp 9`
- `logOf9 = log 9`
- `Squared9 = 9 ** 2`
- `truncateVal = truncate 9.999`
- `roundVal = round 9.999`
- `ceilingVal = ceiling 9.999`
- `floorVal = floor 9.999`
- Also
 - `sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, asinh, acosh, atanh`

EXPLORE THESE

List - Concatenation



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a Haskell interpreter. It displays the following session:

```

WinGHCi
File Edit Actions Tools Help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main      ( haskell-tutorial.hs, interpreted )
Ok, 1 module loaded.
*Main> primeNumber
[3,5,7,11]
*Main> :r
[1 of 1] Compiling Main      ( haskell-tutorial.hs, interpreted )
Ok, 1 module loaded.
*Main> morePrime
[3,5,7,11,13,17,19,23,29]
*Main>

```

On the right is the Notepad++ window, which contains the Haskell code for defining and concatenating lists:

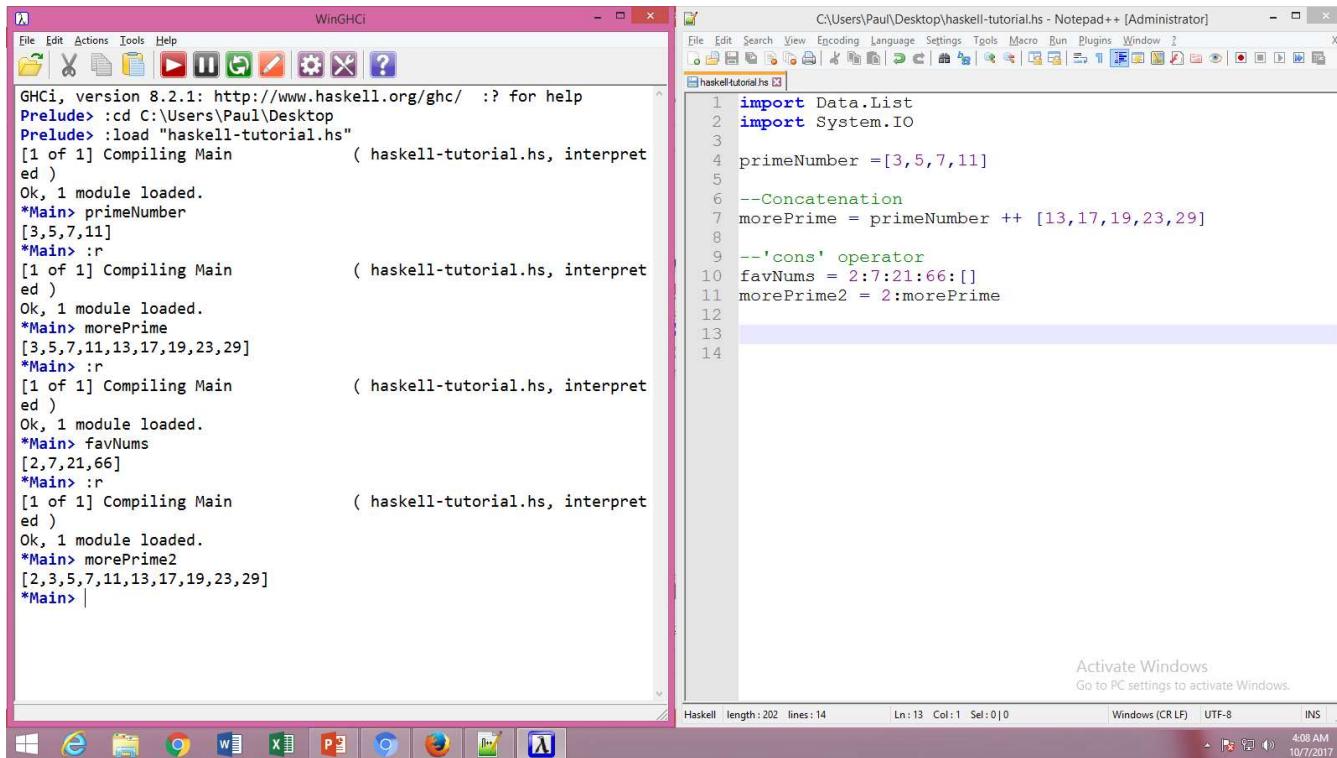
```

C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11] ←
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29] ←
8

```

A blue arrow points from the text "Define a list" to the line `primeNumber =[3,5,7,11]`. Another blue arrow points from the text "Concatenation of two lists" to the line `morePrime = primeNumber ++ [13,17,19,23,29]`.

List – ‘cons’ operator



The image shows a Windows desktop environment with two open windows:

- WinGHCi**: A terminal window showing the Haskell Interactive Component (GHCi) version 8.2.1. The session starts with GHCi, version 8.2.1, and then loads the file "haskell-tutorial.hs". It defines three lists: primeNumber, morePrime, and favNums, and then concatenates them using the '++' operator to produce a final list.
- C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]**: A code editor window containing the Haskell source code "haskell-tutorial.hs". The code imports Data.List and System.IO, defines primeNumber, morePrime, and favNums, and demonstrates the use of the '++' operator for concatenation and the ':r' operator for reading files.

```

WinGHCi
File Edit Actions Tools Help
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs

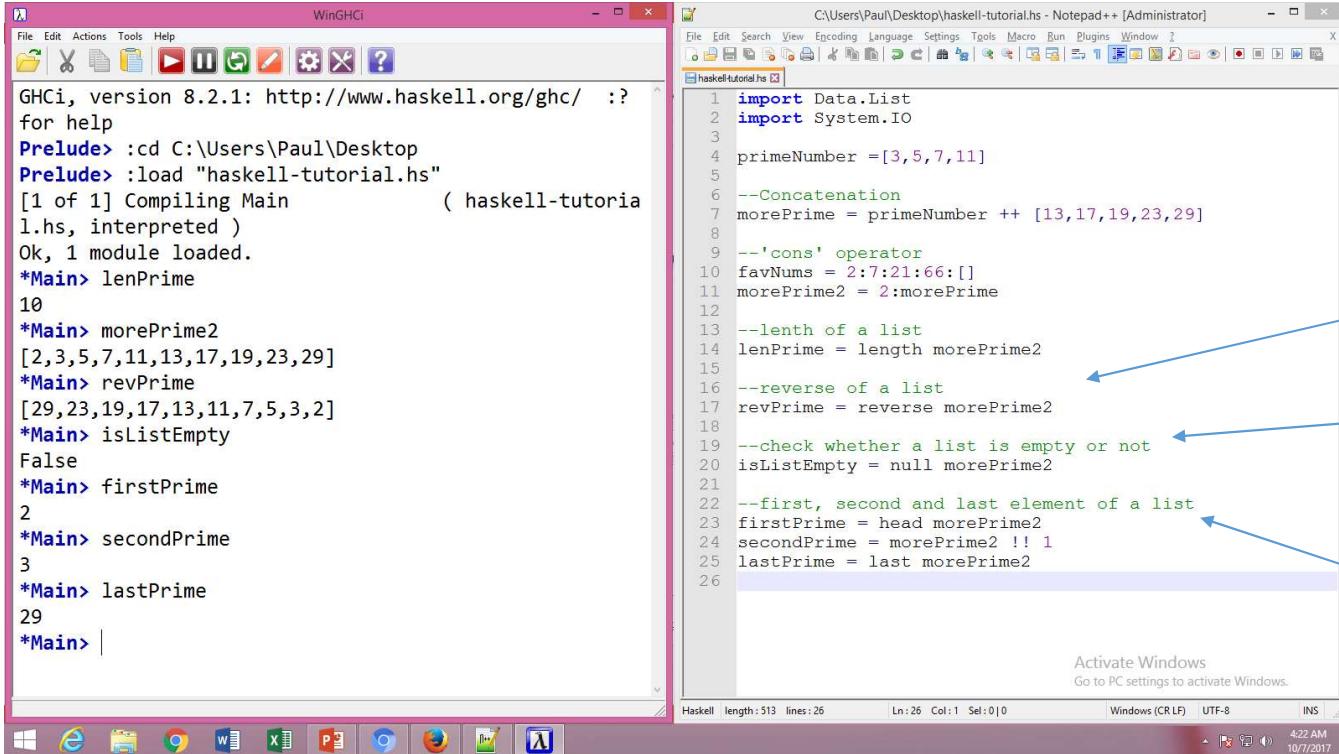
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 -- 'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13
14

Activate Windows
Go to PC settings to activate Windows.

Haskell length: 202 lines: 14 Ln:13 Col:1 Sel:0|0 Windows (CR LF) UTF-8 INS | 4:08 AM 10/7/2017

```

More Operations on List



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a Haskell REPL. It displays a session where various list operations are performed. On the right is a Notepad++ window containing a Haskell script named 'haskell-tutorial.hs'.

```

WinGHCi session transcript:
GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             ( haskell-tutorial.hs, interpreted )
Ok, 1 module loaded.
*Main> lenPrime
10
*Main> morePrime2
[2,3,5,7,11,13,17,19,23,29]
*Main> revPrime
[29,23,19,17,13,11,7,5,3,2]
*Main> isEmpty
False
*Main> firstPrime
2
*Main> secondPrime
3
*Main> lastPrime
29
*Main> |
```

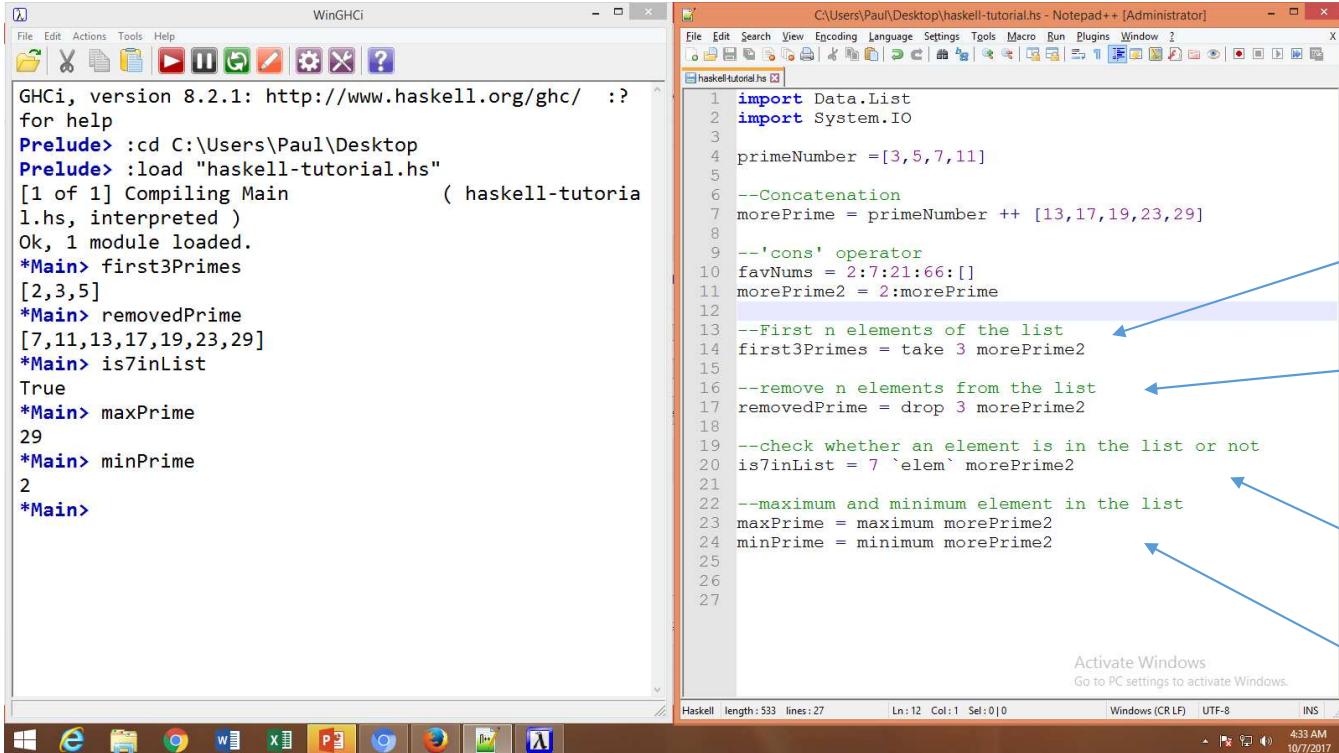
```

haskell-tutorial.hs content:
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --length of a list
14 lenPrime = length morePrime2
15
16 --reverse of a list
17 revPrime = reverse morePrime2
18
19 --check whether a list is empty or not
20 isEmpty = null morePrime2
21
22 --first, second and last element of a list
23 firstPrime = head morePrime2
24 secondPrime = morePrime2 !! 1
25 lastPrime = last morePrime2
26
```

Annotations on the right side of the Notepad++ window:

- reverse**: Points to the line `revPrime = reverse morePrime2`.
- List empty?**: Points to the line `isEmpty = null morePrime2`.
- particular element**: Points to the line `firstPrime = head morePrime2`.

More Operations on List



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a Haskell REPL. It displays the following session:

```

GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> first3Primes
[2,3,5]
*Main> removedPrime
[7,11,13,17,19,23,29]
*Main> is7inList
True
*Main> maxPrime
29
*Main> minPrime
2
*Main>

```

On the right is the Notepad++ window, which contains the Haskell code for the operations shown in the REPL. The code is as follows:

```

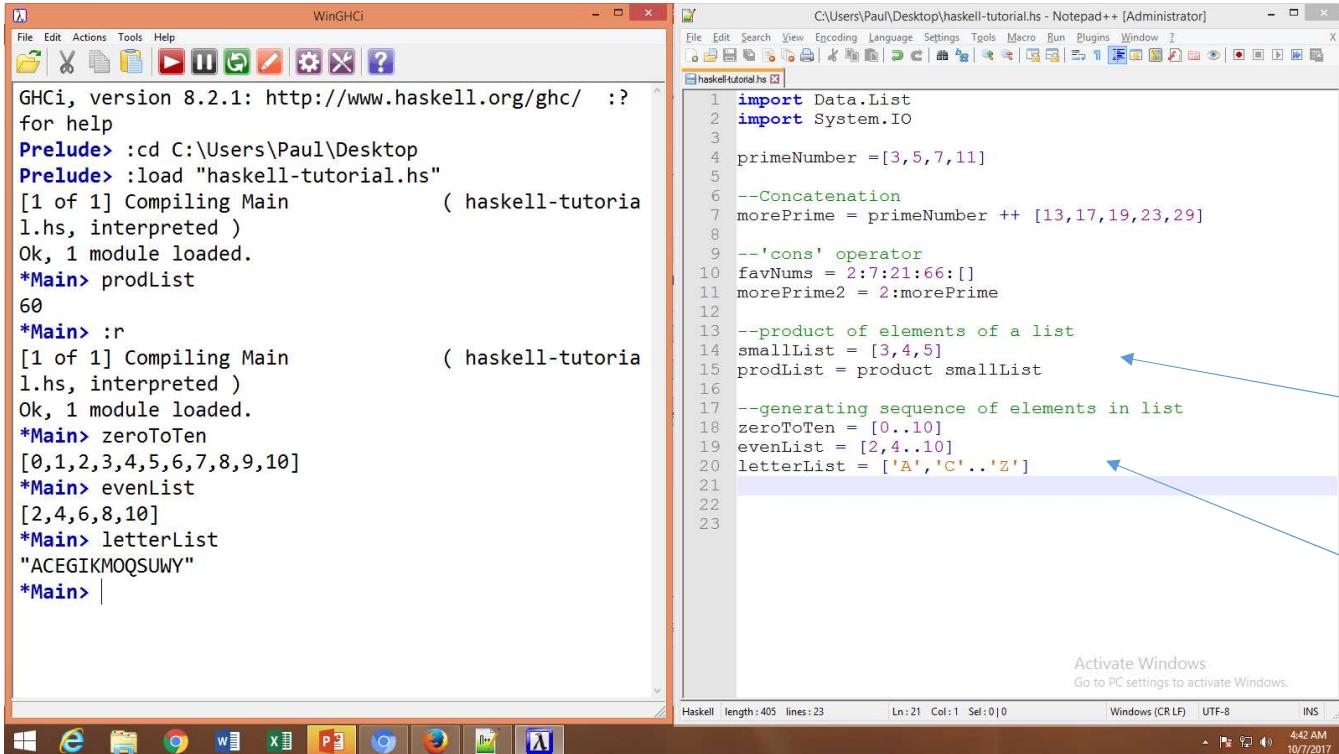
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --First n elements of the list
14 first3Primes = take 3 morePrime2
15
16 --remove n elements from the list
17 removedPrime = drop 3 morePrime2
18
19 --check whether an element is in the list or not
20 is7inList = 7 `elem` morePrime2
21
22 --maximum and minimum element in the list
23 maxPrime = maximum morePrime2
24 minPrime = minimum morePrime2
25
26
27

```

Four red boxes with arrows point from specific text in the code to their respective descriptions:

- An arrow points from the line `first3Primes = take 3 morePrime2` to the box labeled "first n element".
- An arrow points from the line `removedPrime = drop 3 morePrime2` to the box labeled "removing first n element".
- An arrow points from the line `is7inList = 7 `elem` morePrime2` to the box labeled "finding an element".
- An arrow points from the lines `maxPrime = maximum morePrime2` and `minPrime = minimum morePrime2` to the box labeled "max and min".

More Operations on List



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a Haskell REPL. It displays several commands entered and their results:

```

WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> prodList
60
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> zeroToTen
[0,1,2,3,4,5,6,7,8,9,10]
*Main> evenList
[2,4,6,8,10]
*Main> letterList
"ACEGIKMOQSUWY"
*Main>

```

On the right is the Notepad++ window, showing a Haskell script named `haskell-tutorial.hs`. The code demonstrates various list operations:

```

C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs [x]
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --product of elements of a list
14 smallList = [3,4,5]
15 prodList = product smallList
16
17 --generating sequence of elements in list
18 zeroToTen = [0..10]
19 evenList = [2,4..10]
20 letterList = ['A','C'..'Z']
21
22
23

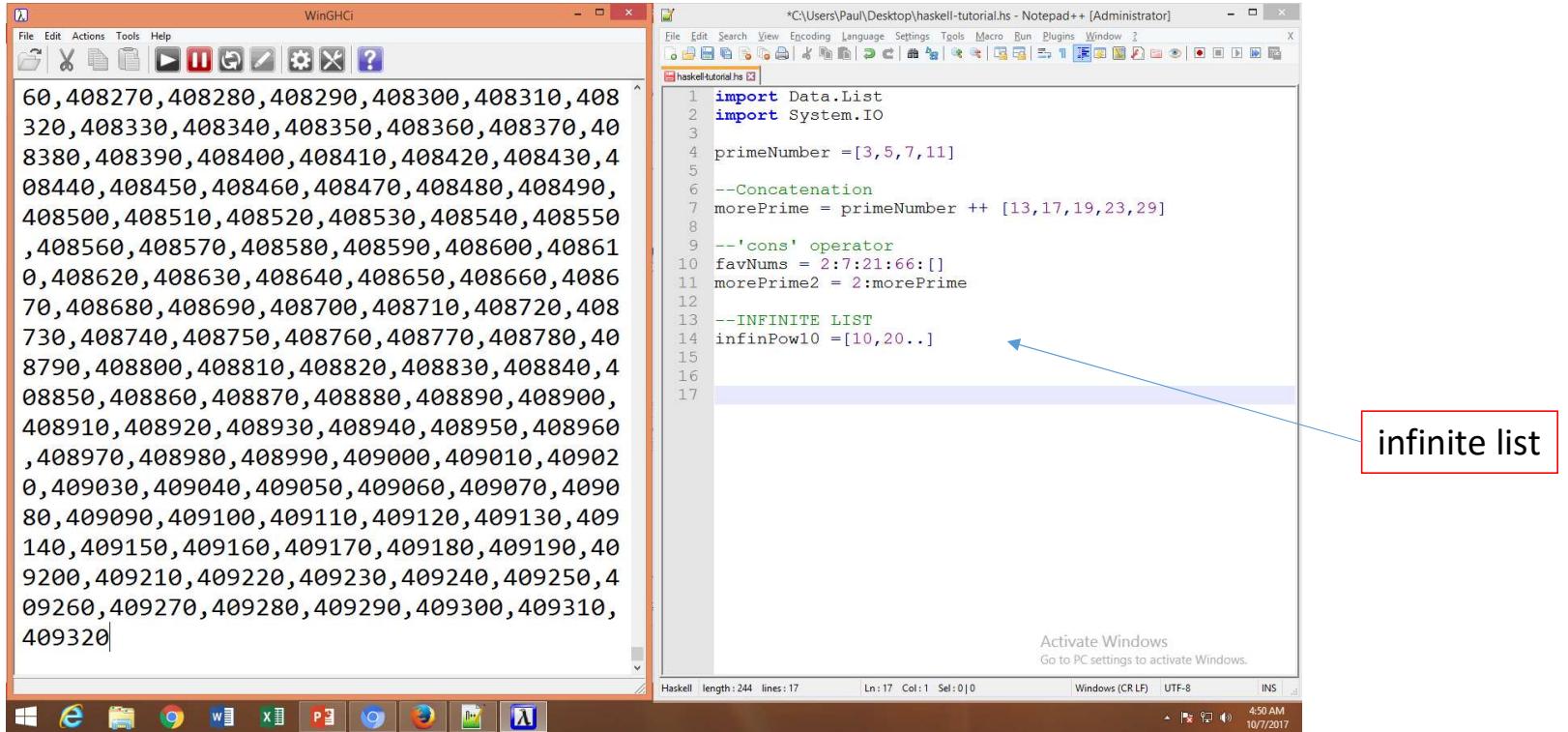
```

Two red boxes with arrows point from the text to the right of the code to the words "product" and "sequence".

product

sequence

More Operations on List



The image shows a Windows desktop environment with two open windows:

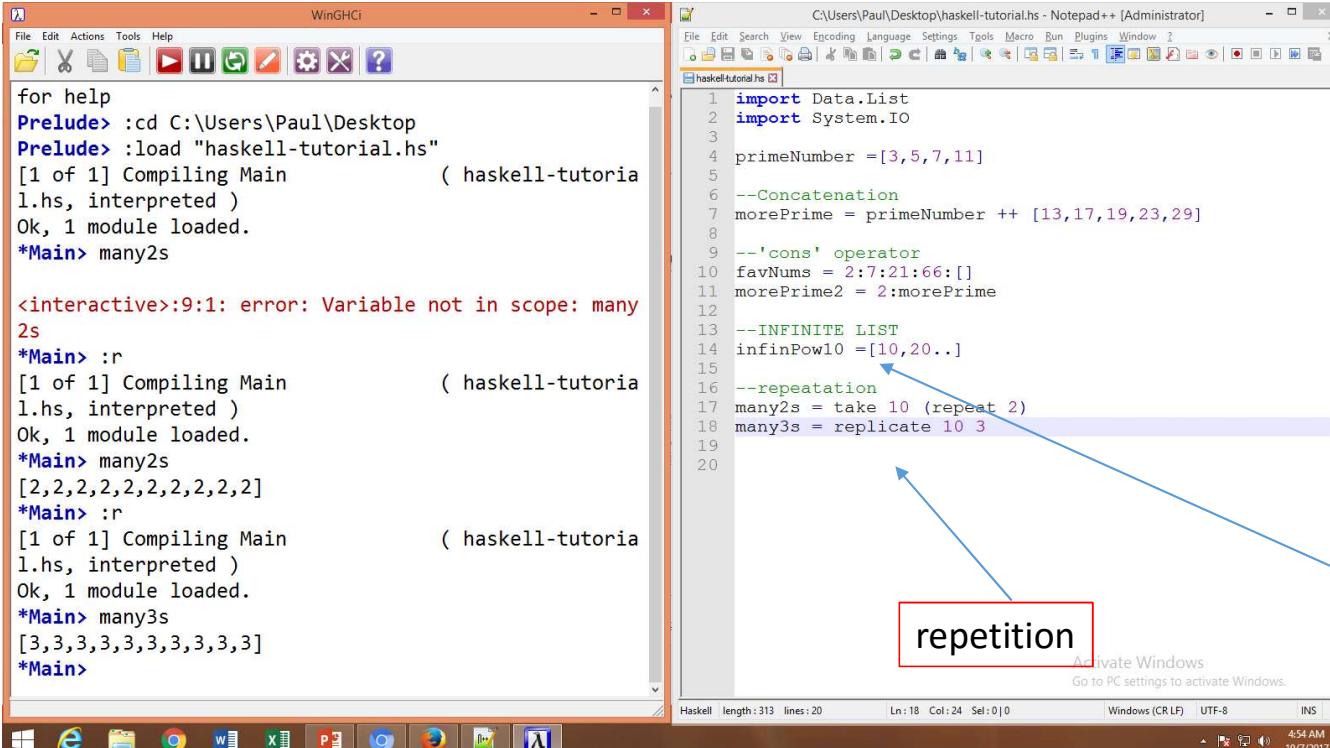
- WinGHCi**: A terminal window showing a long list of integers starting from 60 and increasing by 408 each time, illustrating an infinite list.
- Notepad++**: An IDE window containing Haskell code for creating an infinite list of prime numbers.

```

1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --INFINITE LIST
14 infinPow10 =[10,20..]
15
16
17
  
```

A red box highlights the text "infinite list" in the Notepad++ window, with a blue arrow pointing from it to the corresponding text in the WinGHCi window.

More Operations on List



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a Haskell REPL. It displays the following session:

```

for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             ( haskell-tutoria
l.hs, interpreted )
Ok, 1 module loaded.
*Main> many2s

<interactive>:9:1: error: Variable not in scope: many
2s
*Main> :r
[1 of 1] Compiling Main             ( haskell-tutoria
l.hs, interpreted )
Ok, 1 module loaded.
*Main> many2s
[2,2,2,2,2,2,2,2,2,2]
*Main> :r
[1 of 1] Compiling Main             ( haskell-tutoria
l.hs, interpreted )
Ok, 1 module loaded.
*Main> many3s
[3,3,3,3,3,3,3,3,3,3]
*Main>

```

On the right is the Notepad++ window, showing the code for "haskell-tutorial.hs". The code includes imports for Data.List and System.IO, defines primeNumber, morePrime, and morePrime2, and demonstrates various list operations like concatenation, repetition, and infinite lists.

```

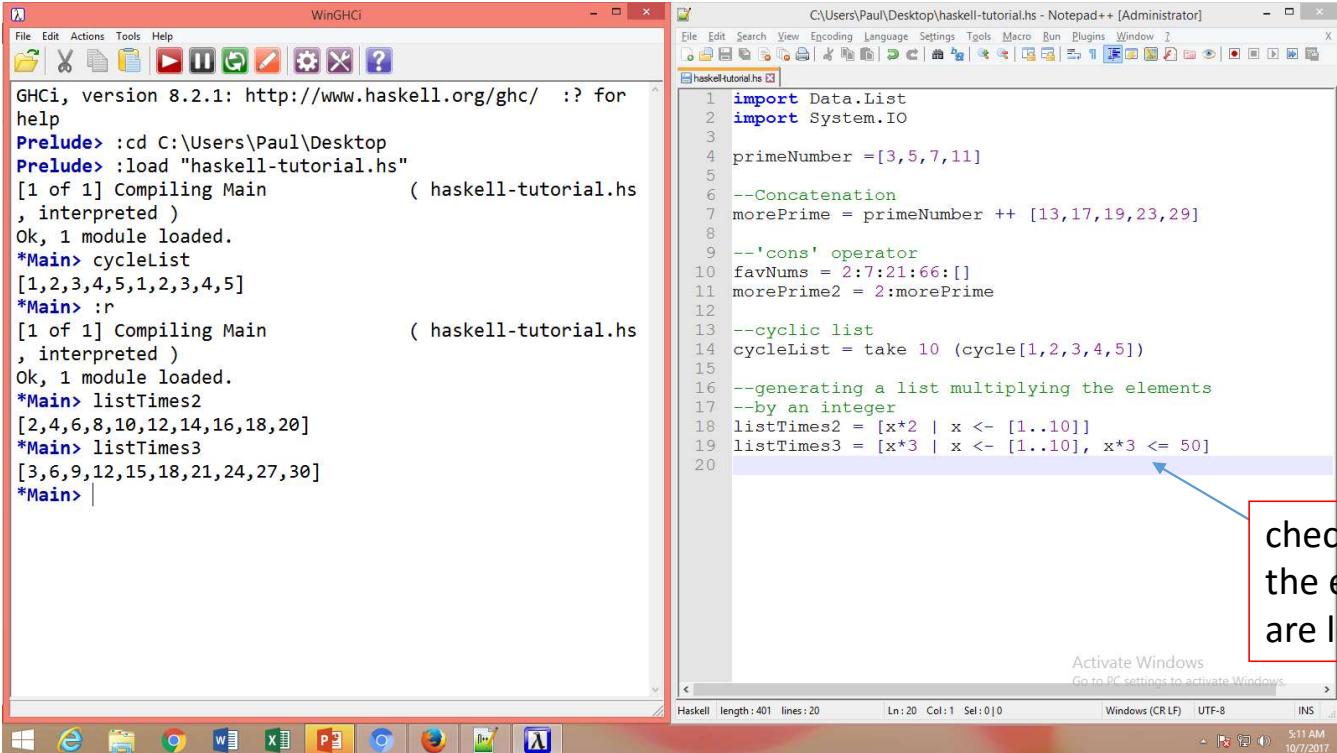
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --INFINITE LIST
14 infinPow10 =[10,20...]
15
16 --repeataiton
17 many2s = take 10 (repeat 2)
18 many3s = replicate 10 3
19
20

```

A red box highlights the word "repetition" in the code, and a blue arrow points from this box to the "repeat" function call in line 17.

One of the examples of advantages of laziness property and functional approach: here, the presence of ***infinite list*** does not affect other expressions/functions in the program

More Operations on List



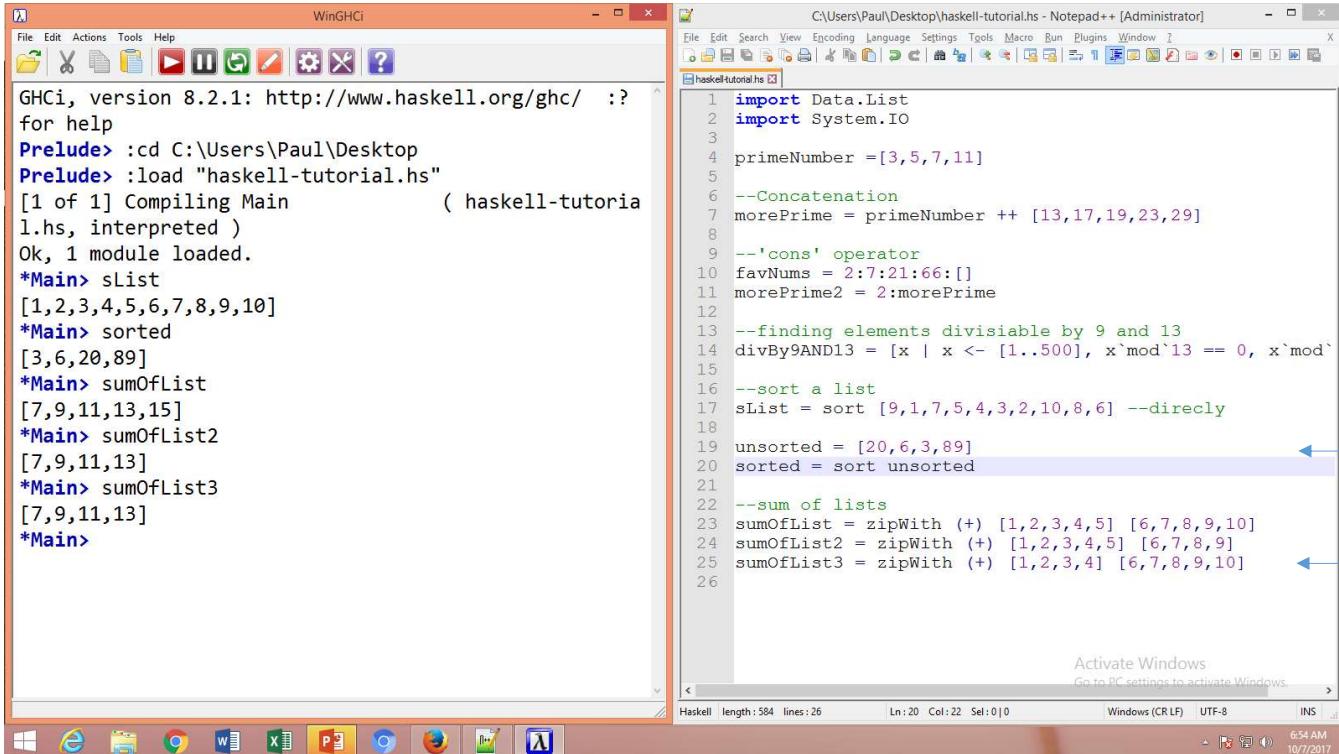
The image shows a Windows desktop environment with two open windows:

- WinGHCi**: A terminal window showing the Haskell Interactive Component (GHCi) version 8.2.1. The user has loaded a file named "haskell-tutorial.hs" and is demonstrating various list operations like concatenation, mapping, and filtering.
- Notepad++ [Administrator]**: An editor window containing the code for "haskell-tutorial.hs". The code includes imports for Data.List and System.IO, defines prime numbers, and uses list comprehensions to generate and filter lists.

A red callout box with a blue arrow points from the text "check whether the element generated are less than 50 or not" to the line of code:

```
listTimes3 = [x*3 | x <- [1..10], x*3 <= 50]
```

More Operations on List



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a Haskell REPL. It displays a session where various list operations are performed, such as creating lists, sorting them, and summing their elements. On the right is a Notepad++ window containing a Haskell script named 'haskell-tutorial.hs'. The script defines several functions for manipulating lists, including sorting and summation. Two red boxes with arrows point from the text to specific parts of the code: one labeled 'sorting' points to the 'sort' function, and another labeled 'summation' points to the 'sumOfList' function.

```

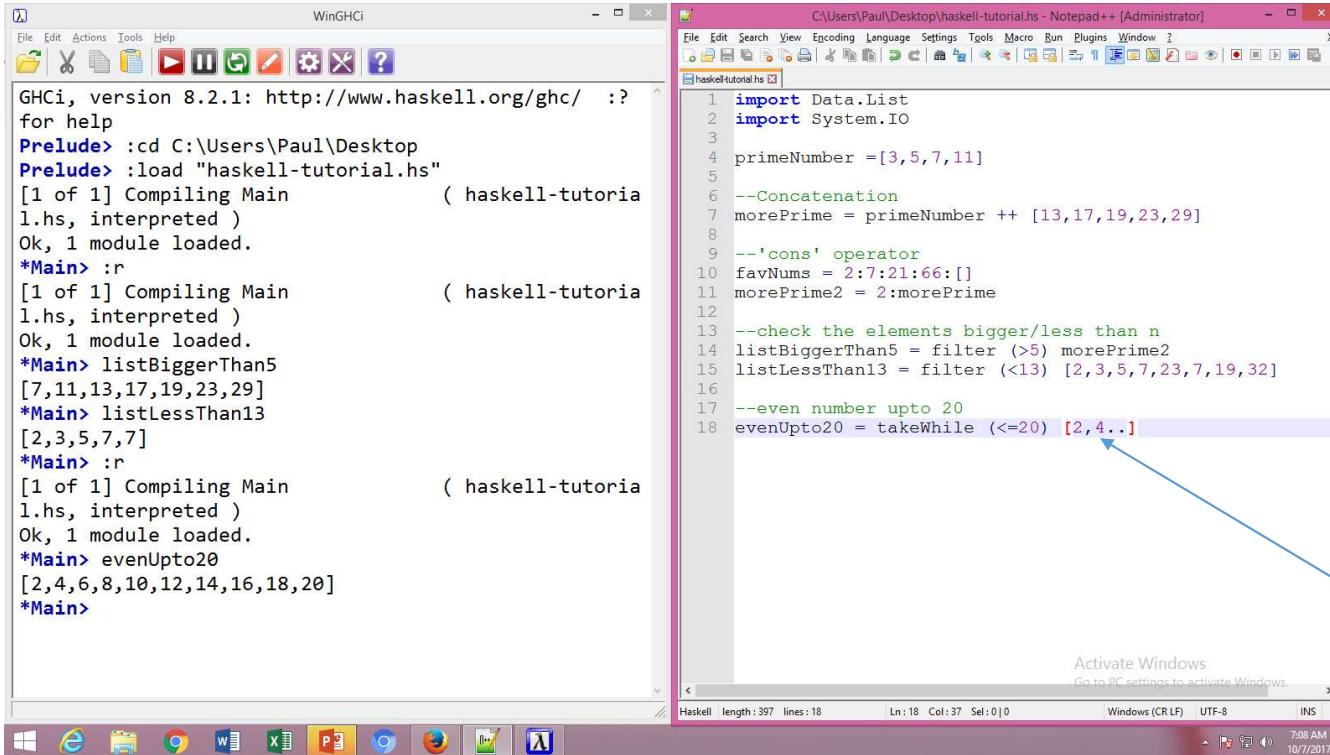
WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             ( haskell-tutorial.hs, interpreted )
Ok, 1 module loaded.
*Main> sList
[1,2,3,4,5,6,7,8,9,10]
*Main> sorted
[3,6,20,89]
*Main> sumOfList
[7,9,11,13,15]
*Main> sumOfList2
[7,9,11,13]
*Main> sumOfList3
[7,9,11,13]
*Main>

C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 primeNumber = [3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --finding elements divisible by 9 and 13
14 divBy9AND13 = [x | x <- [1..500], x`mod`13 == 0, x`mod`9 == 0]
15
16 --sort a list
17 sList = sort [9,1,7,5,4,3,2,10,8,6] --directly
18
19 unsorted = [20,6,3,89]
20 sorted = sort unsorted
21
22 --sum of lists
23 sumOfList = zipWith (+) [1,2,3,4,5] [6,7,8,9,10]
24 sumOfList2 = zipWith (+) [1,2,3,4,5] [6,7,8,9]
25 sumOfList3 = zipWith (+) [1,2,3,4] [6,7,8,9,10]
26

Activate Windows
Go to PC settings to activate Windows.

Haskell length: 584 lines: 26 Ln:20 Col:22 Sel:0|0 Windows (CR LF) UTF-8 INS 6:54 AM 10/7/2017
  
```

More Operations on List



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a REPL for the Haskell programming language. It displays a session where the user is defining a list of prime numbers and then using various list comprehension and filtering functions to create new lists of even numbers up to 20. On the right is a Notepad++ window containing the same Haskell code, with a blue arrow pointing from the text "evenUpto20 = takeWhile (<=20) [2,4..]" in the WinGHCi window to the corresponding line in the Notepad++ window.

```

WinGHCi
File Edit Actions Tools Help
 GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> listBiggerThan5
[7,11,13,17,19,23,29]
*Main> listLessThan13
[2,3,5,7,]
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> evenUpto20
[2,4,6,8,10,12,14,16,18,20]
*Main>

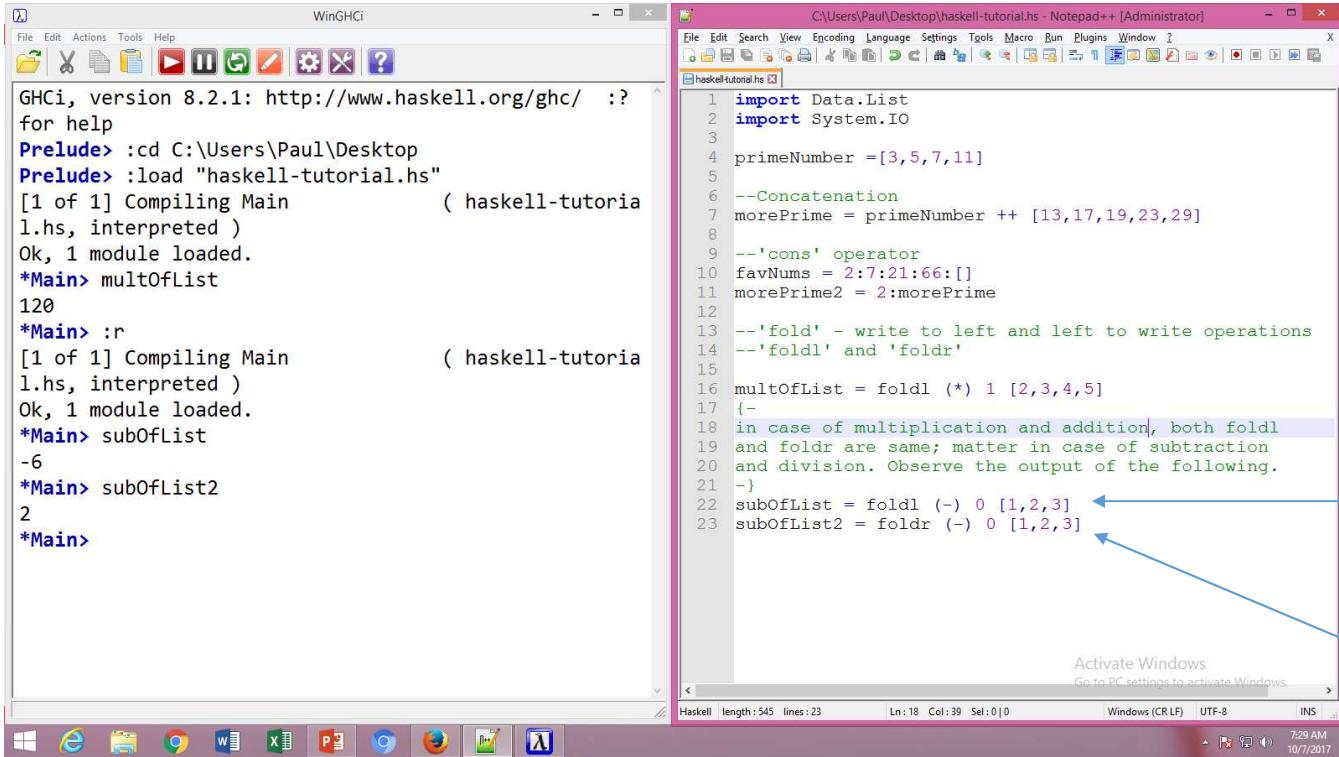
C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --check the elements bigger/less than n
14 listBiggerThan5 = filter (>5) morePrime2
15 listLessThan13 = filter (<13) [2,3,5,7,23,7,19,32]
16
17 --even number upto 20
18 evenUpto20 = takeWhile (<=20) [2,4..]

Activate Windows
Go to PC settings to activate Windows.
Haskell | length: 397 lines: 18 Ln:18 Col:37 Sel:0|0 Windows (CR LF) UTF-8 INS 7:08 AM 10/7/2017

```

another example
of laziness;
although infinite
list, check up to
20

More Operations on List



The image shows two windows side-by-side. On the left is the WinGHCi window, which displays a Haskell session. On the right is the Notepad++ window containing a Haskell script named 'haskell-tutorial.hs'.

WinGHCi Session:

```

GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             ( haskell-tutoria
1.hs, interpreted )
Ok, 1 module loaded.
*Main> multOfList
120
*Main> :r
[1 of 1] Compiling Main             ( haskell-tutoria
1.hs, interpreted )
Ok, 1 module loaded.
*Main> subOfList
-6
*Main> subOfList2
2
*Main>

```

haskell-tutorial.hs Content:

```

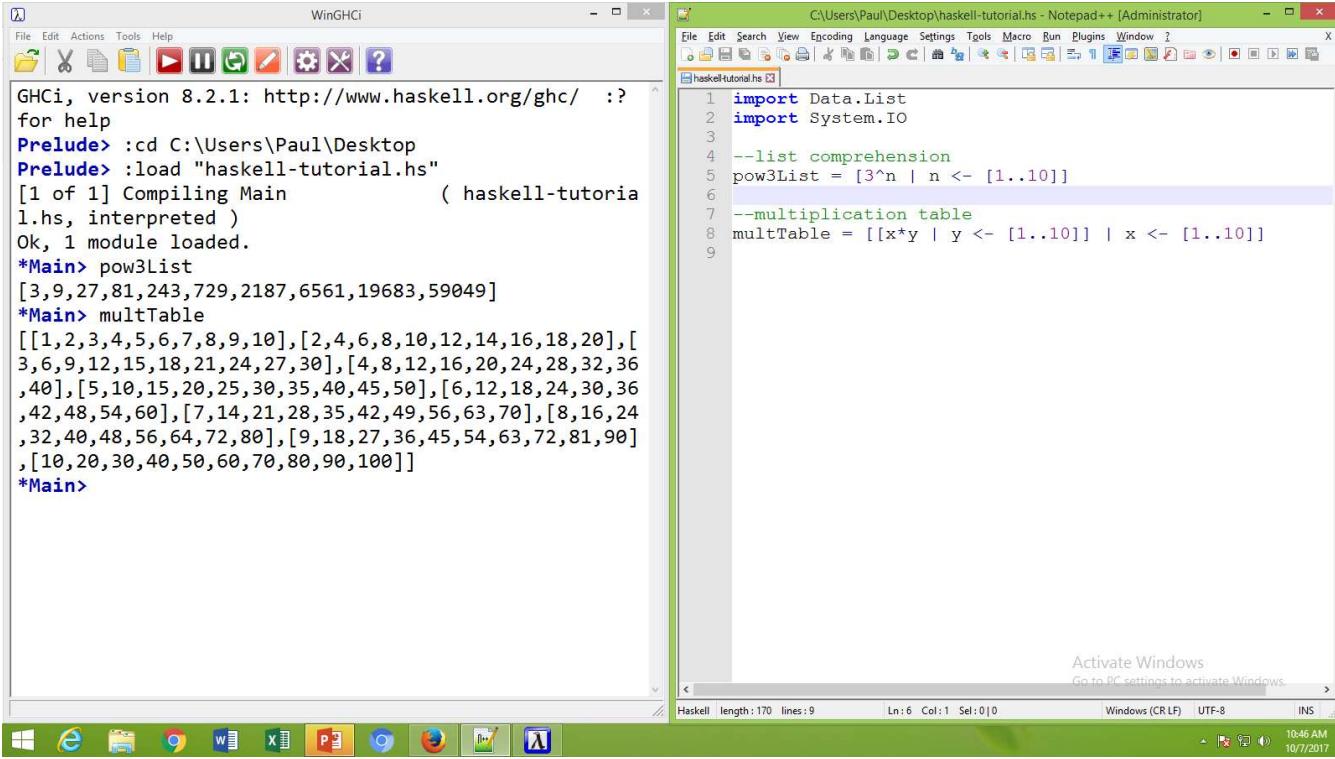
1 import Data.List
2 import System.IO
3
4 primeNumber = [3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --'fold' - write to left and left to write operations
14 --'foldl' and 'foldr'
15
16 multOfList = foldl (*) 1 [2,3,4,5]
17 {-
18 in case of multiplication and addition, both foldl
19 and foldr are same; matter in case of subtraction
20 and division. Observe the output of the following.
21 -}
22 subOfList = foldl (-) 0 [1,2,3]
23 subOfList2 = foldr (-) 0 [1,2,3]

```

Annotations:

- A red box labeled **foldl & foldr** is positioned above the multi-line comment starting at line 18.
- A red box labeled **((0-1)-2)-3)** is positioned to the right of the line `subOfList = foldl (-) 0 [1,2,3]`.
- A red box labeled **1-(2-(3-0))** is positioned to the right of the line `subOfList2 = foldr (-) 0 [1,2,3]`.

More Operations on List



The image shows a Windows desktop environment with two open windows:

- WinGHCi**: A terminal window showing Haskell code execution. The session starts with GHCi version 8.2.1, then changes directory to C:\Users\Paul\Desktop, loads the file "haskell-tutorial.hs", compiles Main, and loads 1 module. It then defines two lists: pow3List and multTable.
- Notepad++ [Administrator]**: An editor window titled "haskell-tutorial.hs" containing the Haskell source code. The code imports Data.List and System.IO, defines a list comprehension for pow3List, and a multiplication table for multTable using list comprehensions.

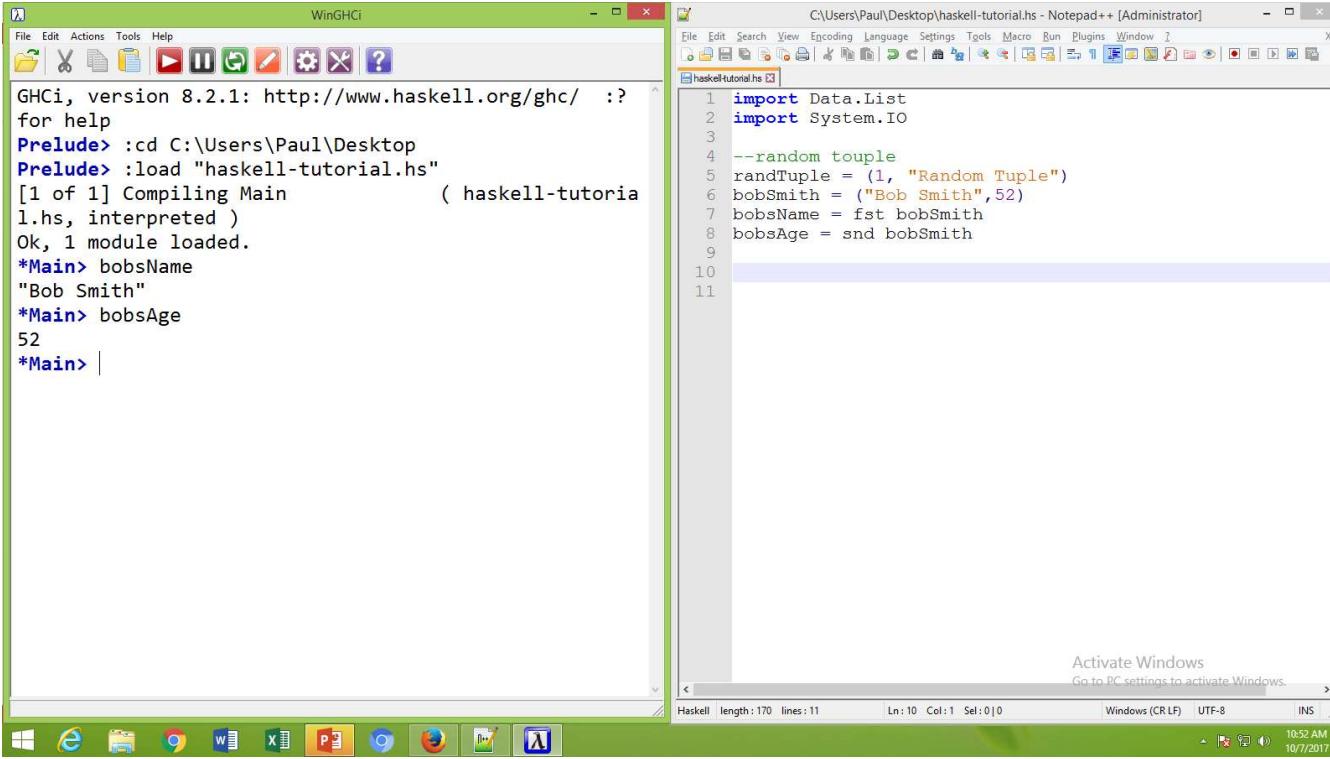
```

WinGHCi
File Edit Actions Tools Help
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 --list comprehension
5 pow3List = [3^n | n <- [1..10]]
6
7 --multiplication table
8 multTable = [[x*y | y <- [1..10]] | x <- [1..10]]
9

Haskell length: 170 lines: 9 Ln: 6 Col: 1 Sel: 0|0 Windows (CR LF) UTF-8 INS
Activate Windows Go to PC settings to activate Windows.
1046 AM 10/7/2017

```

Multiple Data Type



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a Haskell REPL. It displays the following session:

```

WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> bobsName
"Bob Smith"
*Main> bobsAge
52
*Main>

```

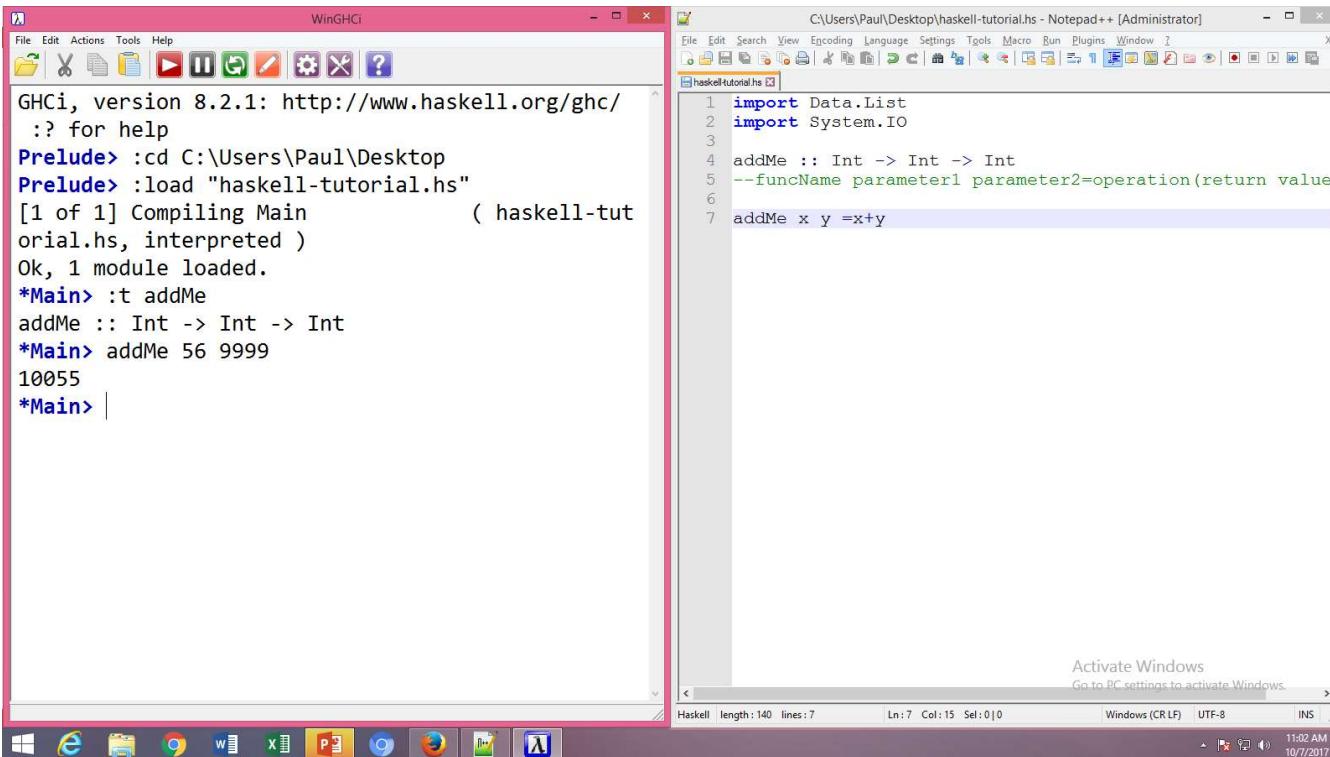
On the right is the Notepad++ window, showing the contents of the file `haskell-tutorial.hs`:

```

C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 --random tuple
5 randTuple = (1, "Random Tuple")
6 bobSmith = ("Bob Smith",52)
7 bobsName = fst bobSmith
8 bobsAge = snd bobSmith
9
10
11

```

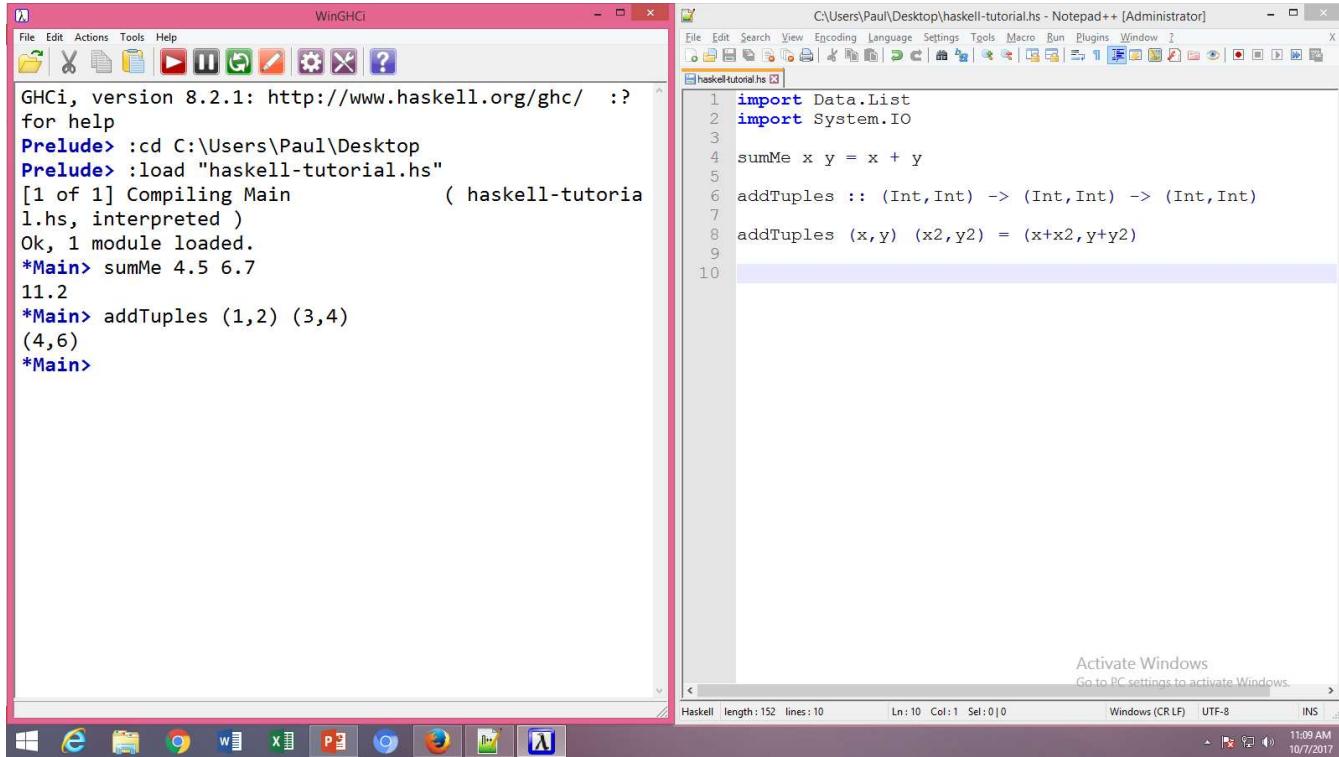
Function Declaration



The image shows a Windows desktop environment with two open windows:

- WinGHCi**: A terminal window showing the Haskell Interactive Component (GHCi) version 8.2.1. The session starts with the GHCi prompt, then changes directory to the user's desktop, loads a file named "haskell-tutorial.hs", and defines a function named "addMe". Finally, it evaluates the expression (addMe 56 9999).
- C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]**: A code editor window displaying the Haskell source code "haskell-tutorial.hs". The code imports Data.List and System.IO, defines a function addMe that takes two Integers and returns an Integer, and includes a multi-line comment explaining the parameters and return type.

User Type Declaration



The image shows a Windows desktop environment with two open windows:

- WinGHCi**: A terminal window showing the Haskell Interactive Component (GHCi) version 8.2.1. The session starts with the GHCi prompt, then changes directory to the user's desktop, loads a file named "haskell-tutorial.hs", compiles the main module, and defines two functions: sumMe and addTuples.

```

GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> sumMe 4.5 6.7
11.2
*Main> addTuples (1,2) (3,4)
(4,6)
*Main>

```

- Notepad++ [Administrator]**: An IDE window titled "haskell-tutorial.hs" containing Haskell code. The code imports Data.List and System.IO, defines sumMe to add two integers, and defines addTuples to add two tuples component-wise.

```

1 import Data.List
2 import System.IO
3
4 sumMe x y = x + y
5
6 addTuples :: (Int,Int) -> (Int,Int) -> (Int,Int)
7
8 addTuples (x,y) (x2,y2) = (x+x2,y+y2)
9
10

```



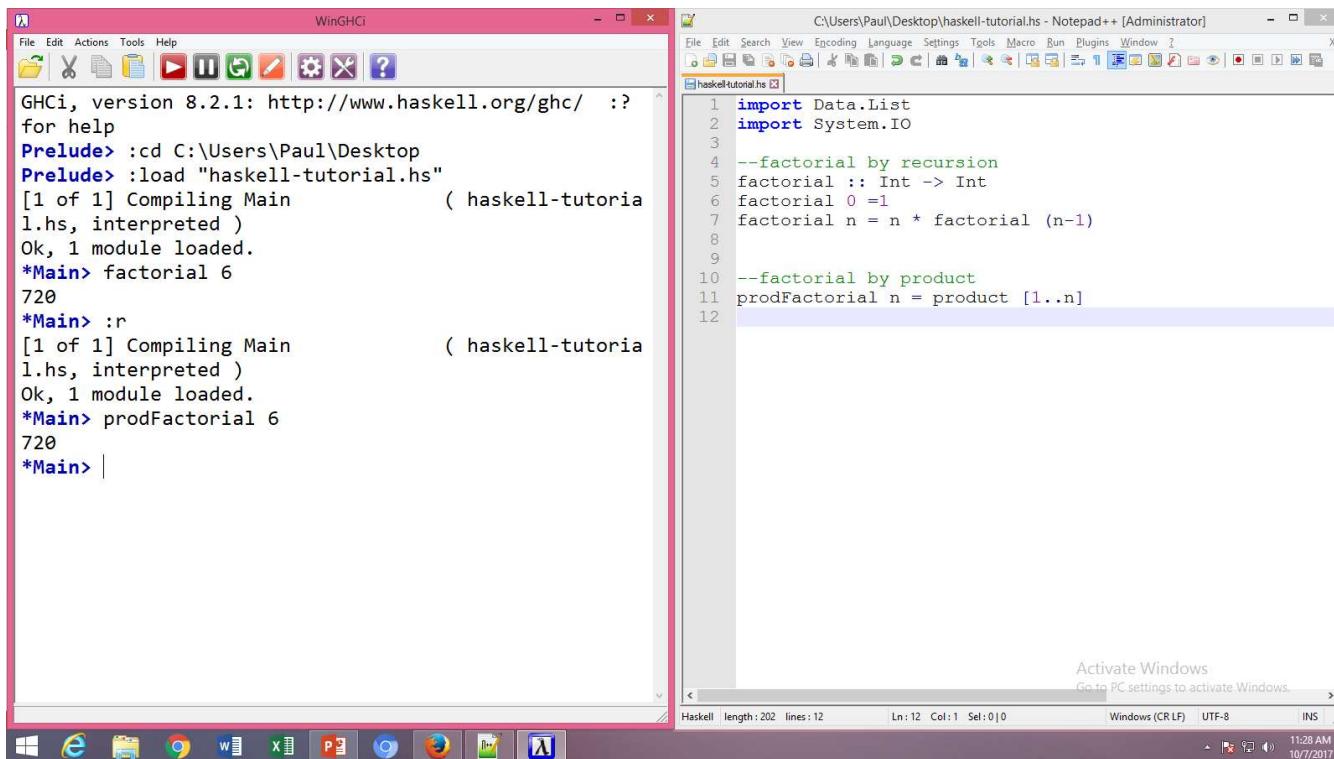
User Type Declaration

The screenshot shows a Windows desktop environment. On the left, the 'WinGHCi' window displays a Haskell interactive session. The session starts with the GHCi version (8.2.1) and its URL. It then shows directory changes, loading of a file ('haskell-tutorial.hs'), compilation of a module ('Main'), and execution of functions like 'sumMe' and 'addTuples'. It also demonstrates type inference with 'whatAge' and its values for different ages. On the right, the 'Notepad++' window shows the source code for 'haskell-tutorial.hs'. The code imports 'Data.List' and 'System.IO', defines a function 'whatAge' that takes an integer and returns a string, and provides specific implementations for ages 16, 18, 21, and a general case for 'x'.

```
WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> sumMe 4.5 6.7
11.2
*Main> addTuples (1,2) (3,4)
(4,6)
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> whatAge 18
"You Can Vote"
*Main> whatAge 16
"You Can Drive"
*Main> whatAge 67
"Nothing Important"
*Main> |
```

```
C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 whatAge :: Int ->String
5
6 whatAge 16 = "You Can Drive"
7 whatAge 18 = "You Can Vote"
8 whatAge 21 = "You are Adult"
9 whatAge x = "Nothing Important"
10
11
```

Factorial (by recursion and by product)



The image shows a Windows desktop environment with two open windows:

- WinGHCi**: A terminal window showing the GHCi interpreter version 8.2.1. It displays the loading of a module named "haskell-tutorial.hs" and the evaluation of the factorial function at the prompt *Main> factorial 6, which returns 720.
- Notepad++ [Administrator]**: An IDE window titled "haskell-tutorial.hs". It contains Haskell code for calculating factorial using both recursion and product. The code includes imports for Data.List and System.IO, and defines two functions: factorial (using recursion) and prodFactorial (using product).

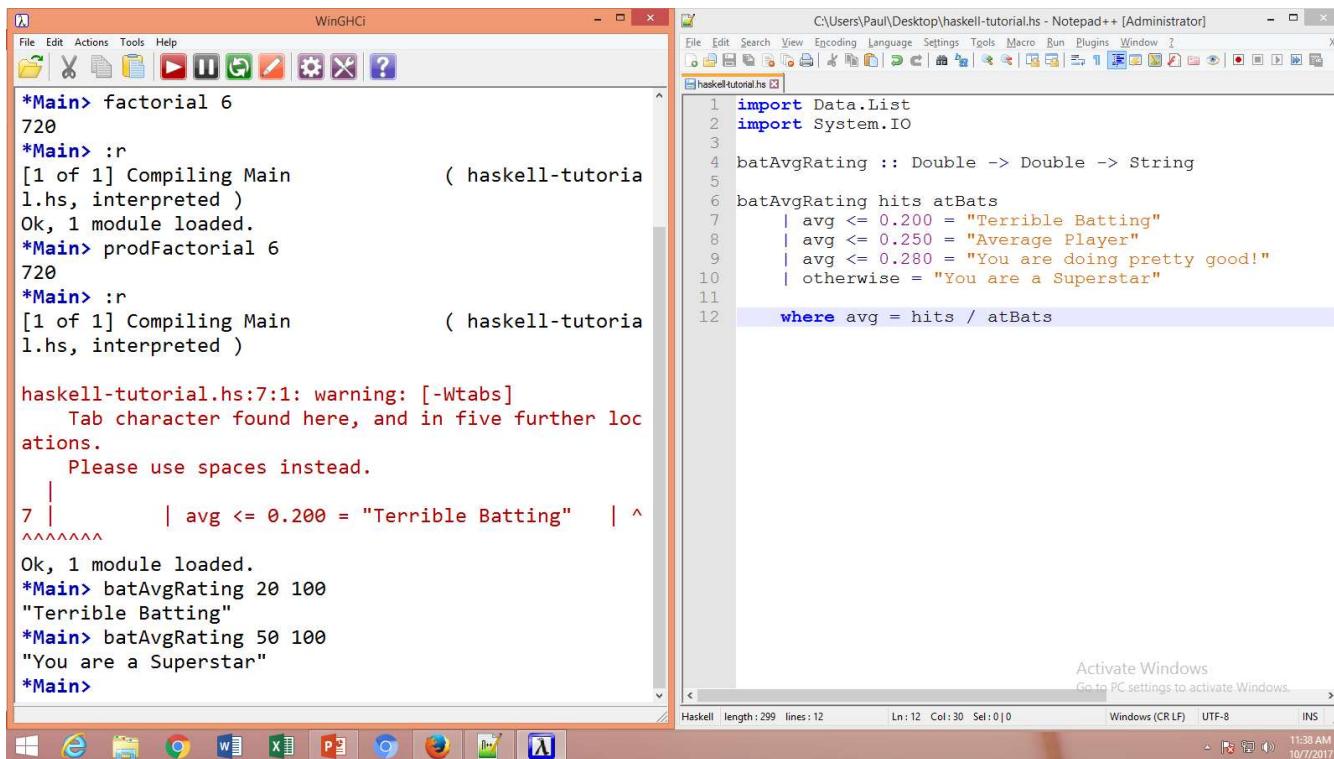
```

WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/  ?:?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> factorial 6
720
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> prodFactorial 6
720
*Main> |
```

```

C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 --factorial by recursion
5 factorial :: Int -> Int
6 factorial 0 = 1
7 factorial n = n * factorial (n-1)
8
9
10 --factorial by product
11 prodFactorial n = product [1..n]
12
```

Guard (where clause)



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a Haskell REPL. It displays the following session:

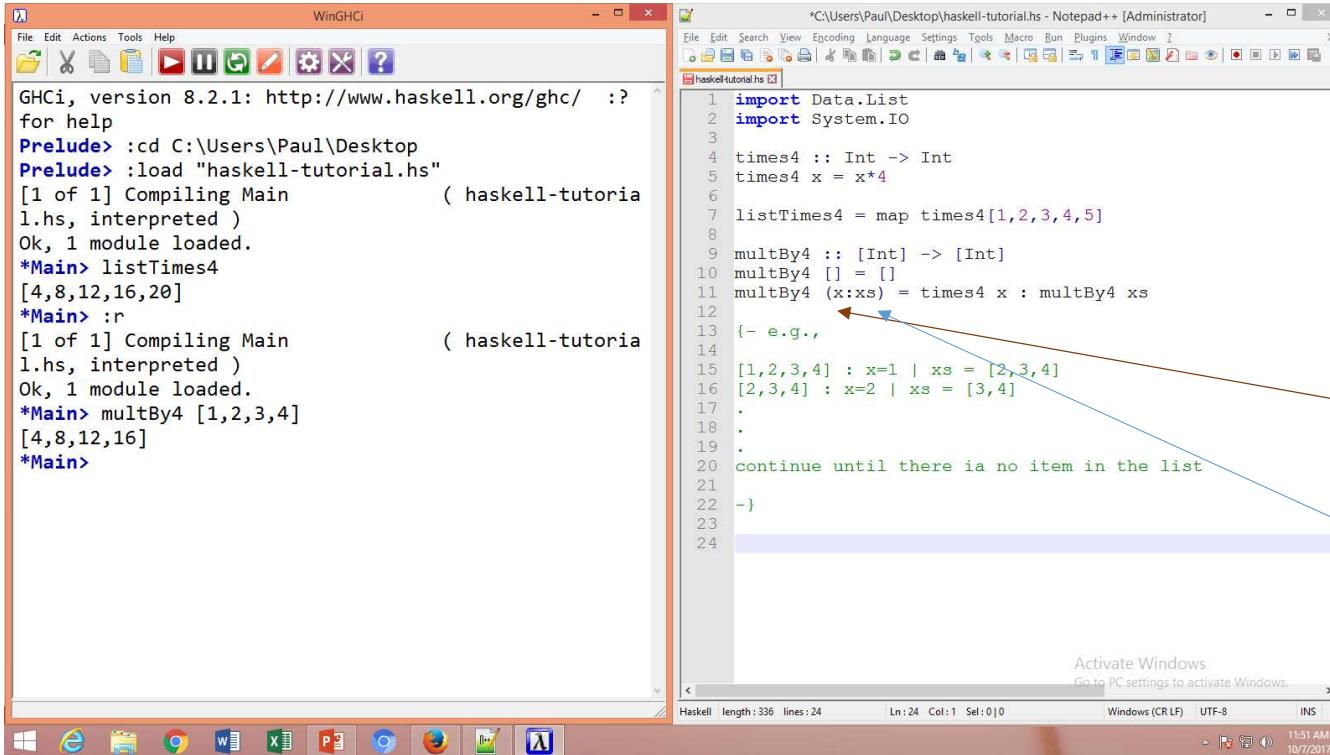
```
*Main> factorial 6
720
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutoria
l.hs, interpreted )
Ok, 1 module loaded.
*Main> prodFactorial 6
720
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutoria
l.hs, interpreted )

haskell-tutorial.hs:7:1: warning: [-Wtabs]
  Tab character found here, and in five further loc
ations.
  Please use spaces instead.
  |           | avg <= 0.200 = "Terrible Batting"    | ^
~~~~~
Ok, 1 module loaded.
*Main> batAvgRating 20 100
"Terrible Batting"
*Main> batAvgRating 50 100
"You are a Superstar"
*Main>
```

On the right is the Notepad++ window, showing the Haskell code for the `batAvgRating` function:

```
1 import Data.List
2 import System.IO
3
4 batAvgRating :: Double -> Double -> String
5
6 batAvgRating hits atBats
7   | avg <= 0.200 = "Terrible Batting"
8   | avg <= 0.250 = "Average Player"
9   | avg <= 0.280 = "You are doing pretty good!"
10  | otherwise = "You are a Superstar"
11
12 where avg = hits / atBats
```

Higher Order Functions



The image shows two windows side-by-side. On the left is the WinGHCi window, which is a Haskell interpreter. It displays the following session:

```

WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> listTimes4
[4,8,12,16,20]
*Main> :r
[1 of 1] Compiling Main             ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> multBy4 [1,2,3,4]
[4,8,12,16]
*Main>

```

On the right is the Notepad++ window, showing the contents of a file named 'haskell-tutorial.hs'. The code defines two functions: 'listTimes4' and 'multBy4'. A callout box with an arrow points from the explanatory text in the red box below to the 'xs' parameter in the 'multBy4' function definition.

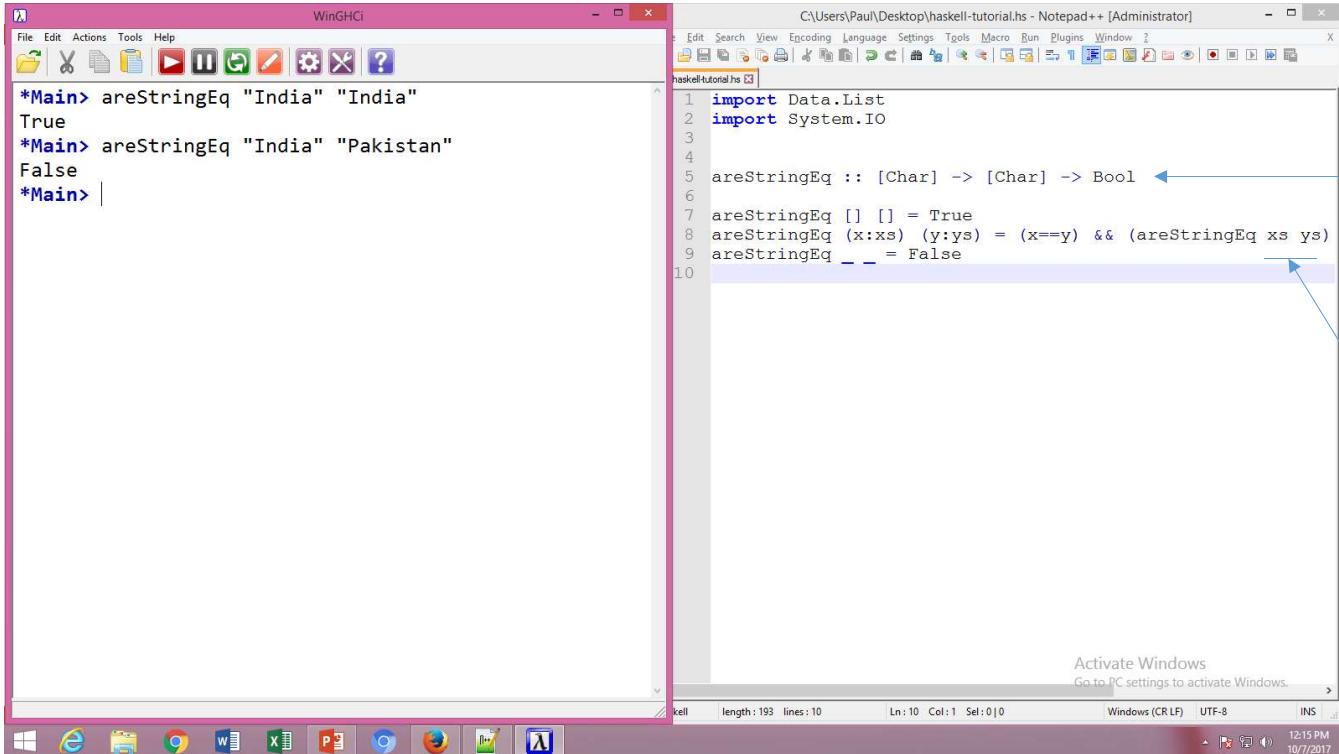
```

*C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 times4 :: Int -> Int
5 times4 x = x^4
6
7 listTimes4 = map times4 [1,2,3,4,5]
8
9 multBy4 :: [Int] -> [Int]
10 multBy4 [] = []
11 multBy4 (x:xs) = times4 x : multBy4 xs
12
13 {- e.g.,
14
15 [1,2,3,4] : x=1 | xs = [2,3,4]
16 [2,3,4] : x=2 | xs = [3,4]
17 .
18 .
19
20 continue until there ia no item in the list
21
22 -}
23
24

```

you don't know how many items in the list
 Beforehand;
 x represents first element in the list, and xs represents remaining elements of the list

Higher Order Functions



The image shows two windows side-by-side. On the left is the WinGHCi window, which contains a command-line interface for the Haskell interpreter. It shows the following interaction:

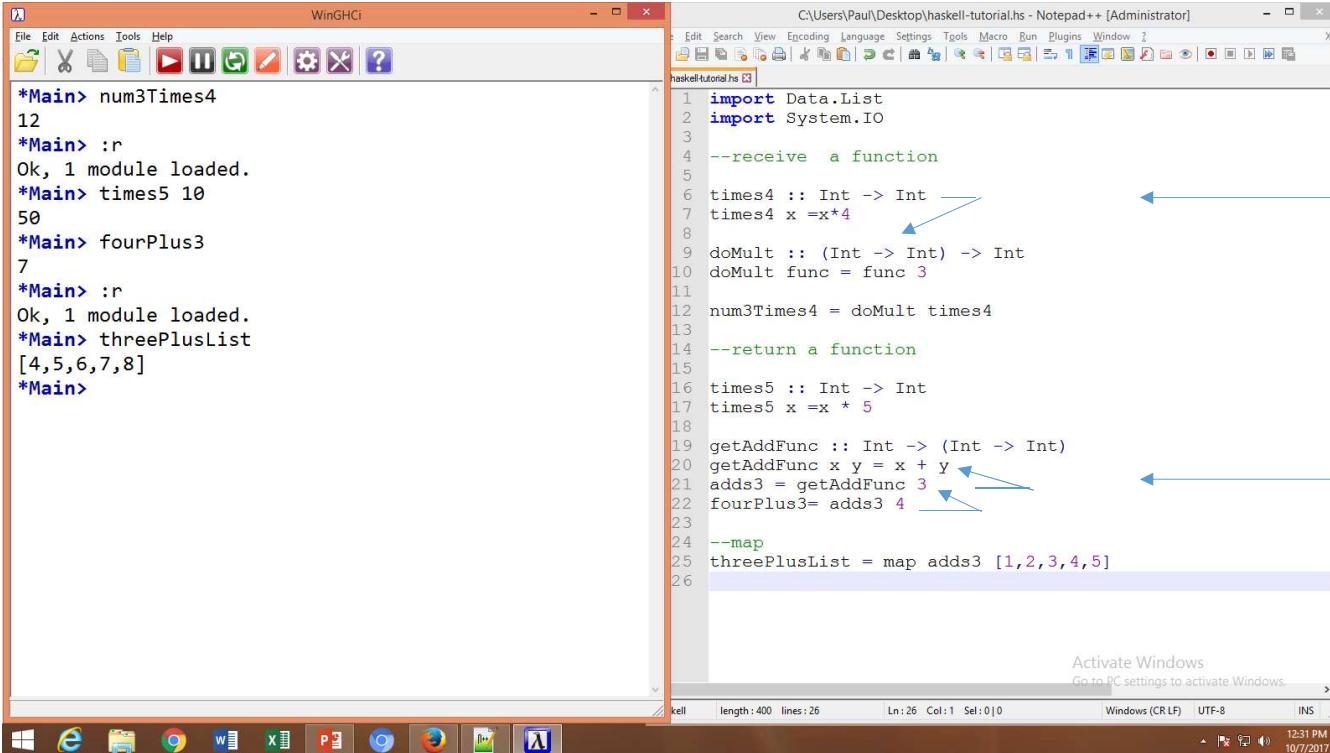
```
*Main> areStringEq "India" "India"
True
*Main> areStringEq "India" "Pakistan"
False
*Main>
```

On the right is the Notepad++ window, which contains the Haskell code for the `areStringEq` function. The code is as follows:

```
1 import Data.List
2 import System.IO
3
4 areStringEq :: [Char] -> [Char] -> Bool
5
6 areStringEq [] [] = True
7 areStringEq (x:xs) (y:ys) = (x==y) && (areStringEq xs ys)
8 areStringEq _ _ = False
```

A red box labeled "return type" has an arrow pointing to the type signature `areStringEq :: [Char] -> [Char] -> Bool`. A blue arrow points from this red box to another red box labeled "remaining" located at the bottom right of the Notepad++ window.

Receive and Return a Function



The screenshot shows two windows side-by-side. On the left is the WinGHCi window, which contains a Haskell REPL session. On the right is the Notepad++ window containing the source code for `haskell-tutorial.hs`.

WinGHCi Session:

```
*Main> num3Times4
12
*Main> :r
Ok, 1 module loaded.
*Main> times5 10
50
*Main> fourPlus3
7
*Main> :r
Ok, 1 module loaded.
*Main> threePlusList
[4,5,6,7,8]
*Main>
```

haskell-tutorial.hs Content:

```
1 import Data.List
2 import System.IO
3
4 --receive a function
5
6 times4 :: Int -> Int
7 times4 x = x^4
8
9 doMult :: (Int -> Int) -> Int
10 doMult func = func 3
11
12 num3Times4 = doMult times4
13
14 --return a function
15
16 times5 :: Int -> Int
17 times5 x = x * 5
18
19 getAddFunc :: Int -> (Int -> Int)
20 getAddFunc x y = x + y
21 add3 = getAddFunc 3
22 fourPlus3 = add3 4
23
24 --map
25 threePlusList = map add3 [1,2,3,4,5]
26
```

Annotations with arrows point from specific code snippets to the corresponding text in the WinGHCi session:

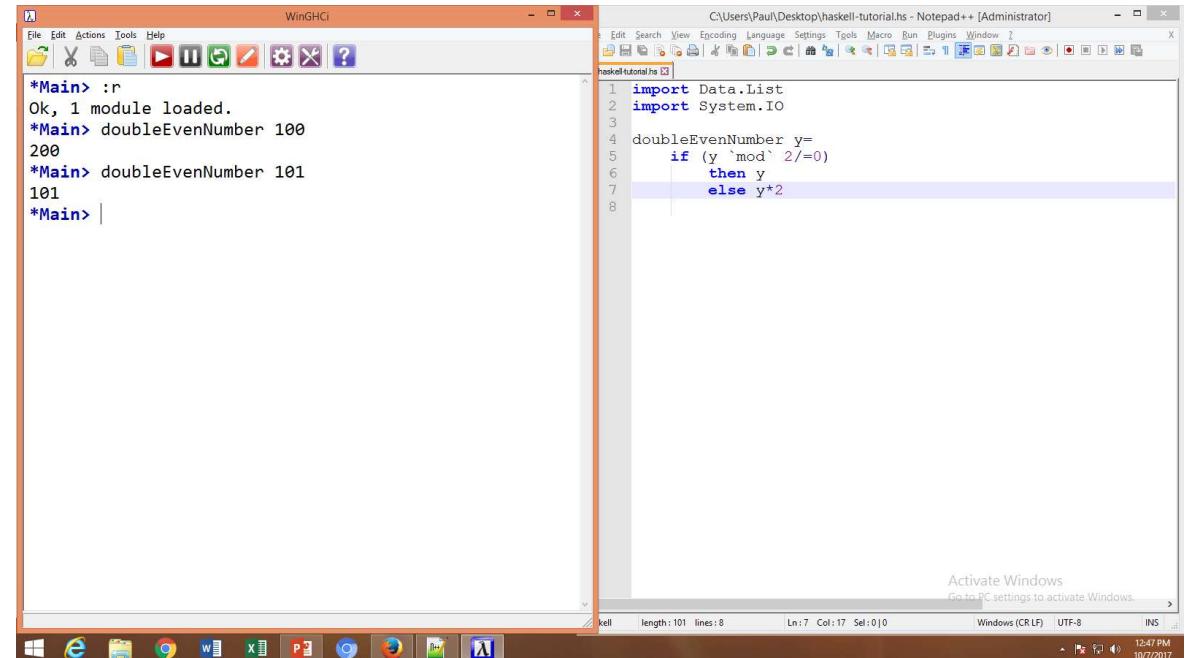
- An arrow points from the line `times4 :: Int -> Int` in the code to the command `num3Times4` in the WinGHCi session, with the word "receive" in a red box to its right.
- An arrow points from the line `getAddFunc :: Int -> (Int -> Int)` in the code to the command `getAddFunc` in the WinGHCi session, with the word "return" in a red box to its right.

Other Operators

➤ Comparison

- < --less than
- > --greater than
- <= --less than equal to
- >= --greater than equal to
- == --equal to

Example



The screenshot shows a Windows desktop environment. On the left, there is a terminal window titled "WinGHCi" which displays Haskell code and its output. On the right, there is a code editor window titled "C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]" containing Haskell code.

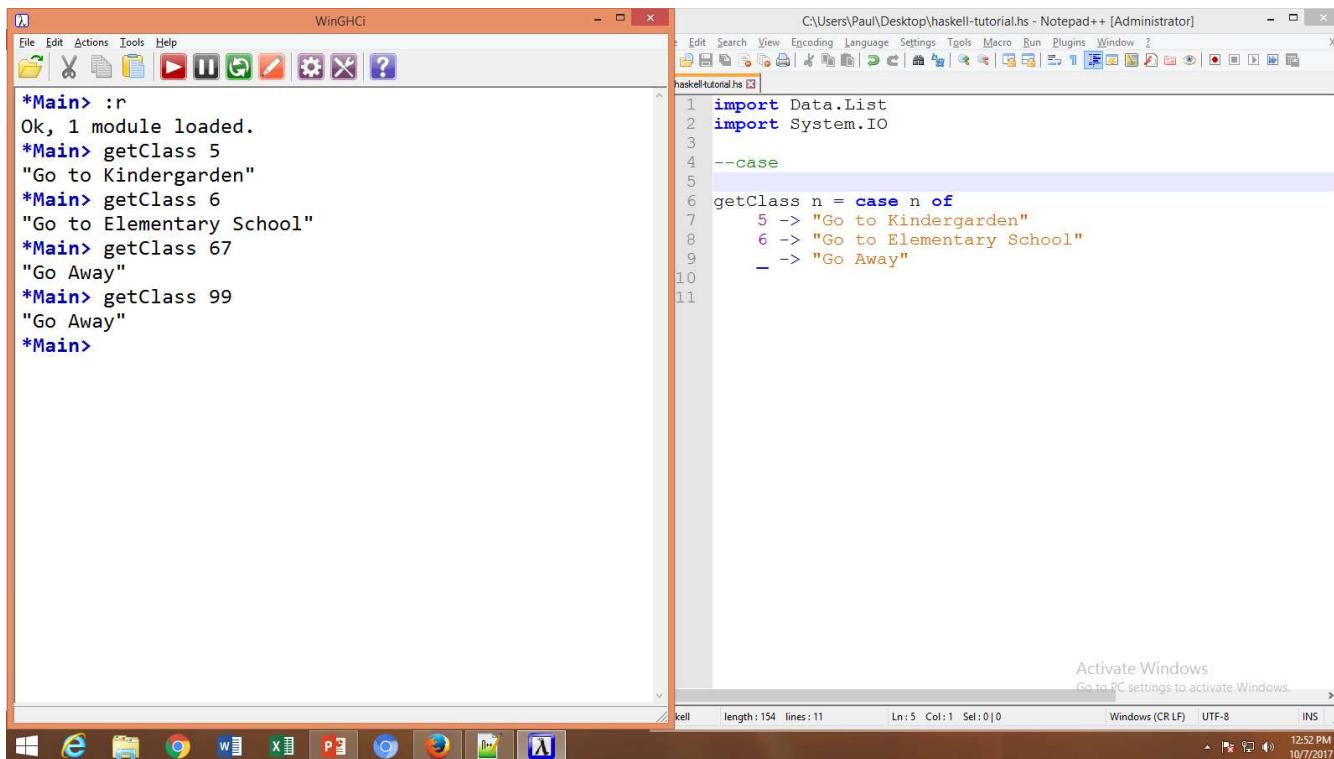
```
*Main> :r
Ok, 1 module loaded.
*Main> doubleEvenNumber 100
200
*Main> doubleEvenNumber 101
101
*Main> |
```

```
1 import Data.List
2 import System.IO
3
4 doubleEvenNumber y=
5   if (y `mod` 2/=0)
6     then y
7     else y*2
```

➤ Logical

- && --AND
- || --OR
- not --NOT

Case



The screenshot shows a Windows desktop environment with two open windows:

- WinGHCi**: A terminal window showing Haskell code execution. The session starts with `*Main> :r`, followed by several calls to `getClass` with values 5, 6, and 67, which return strings like "Go to Kindergarten", "Go to Elementary School", and "Go Away".
- Notepad++ [Administrator]**: An IDE window titled "haskell-tutorial.hs" containing Haskell code. The code imports Data.List and System.IO, defines a function `getClass` using pattern matching, and includes a multi-line string.

```

*Main> :r
Ok, 1 module loaded.
*Main> getClass 5
"Go to Kindergarten"
*Main> getClass 6
"Go to Elementary School"
*Main> getClass 67
"Go Away"
*Main> getClass 99
"Go Away"
*Main>

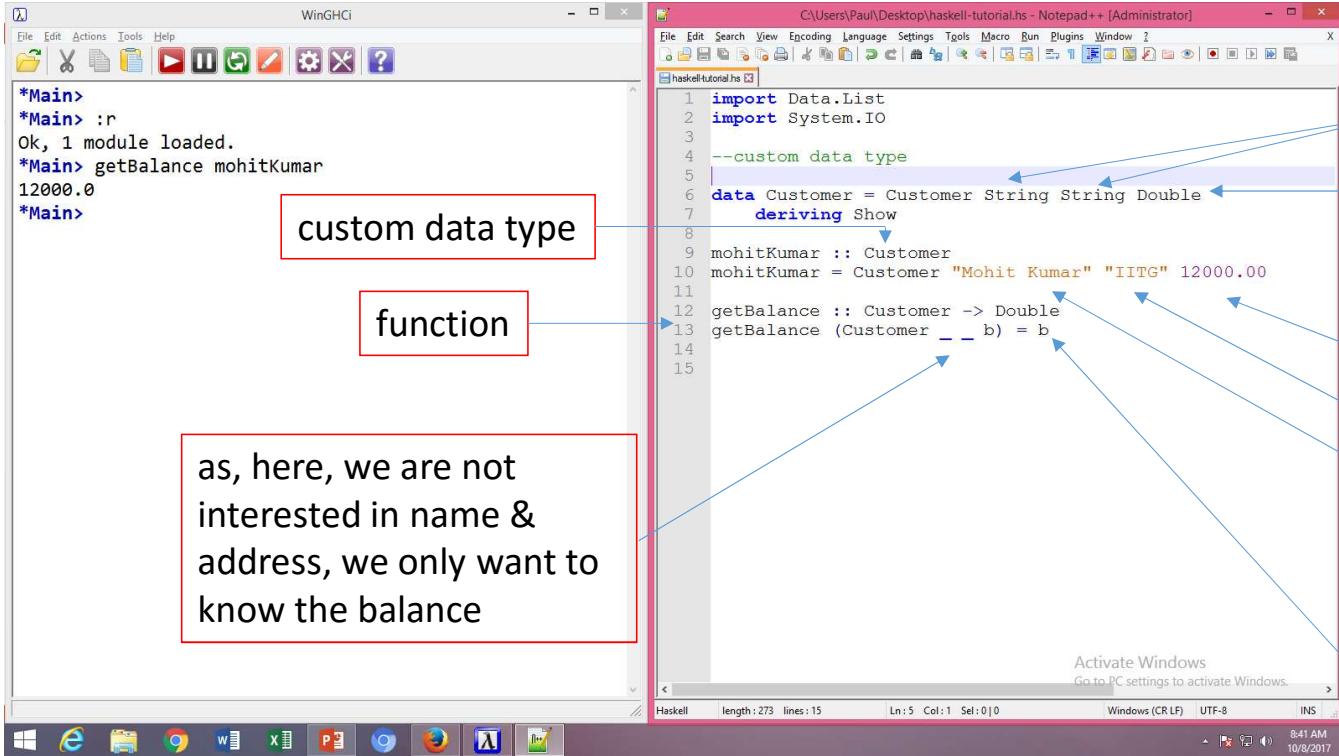
```

```

1 import Data.List
2 import System.IO
3
4 --case
5
6 getClass n = case n of
7   5 -> "Go to Kindergarten"
8   6 -> "Go to Elementary School"
9   _ -> "Go Away"
10
11

```

Custom Data Type



The image shows two windows side-by-side. On the left is WinGHCi, a Haskell REPL, displaying the following session:

```
*Main>
*Main> :r
Ok, 1 module loaded.
*Main> getBalance mohitKumar
12000.0
*Main>
```

Annotations in WinGHCi:

- A red box labeled "custom data type" points to the line `data Customer = Customer String String Double`.
- A red box labeled "function" points to the line `getBalance :: Customer -> Double`.
- A red box containing the text "as, here, we are not interested in name & address, we only want to know the balance" is positioned below the WinGHCi window.

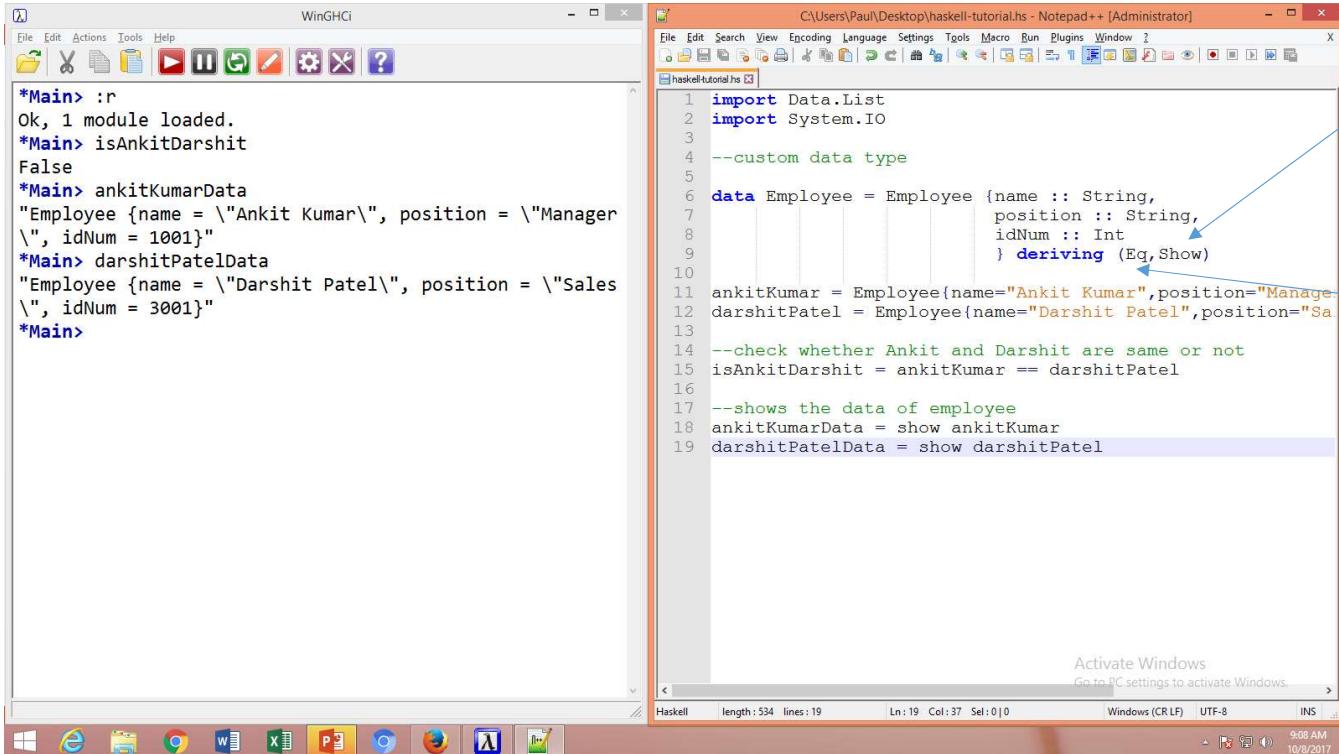
On the right is Notepad++ displaying a Haskell file named `haskellTutorial.hs`:

```
1 import Data.List
2 import System.IO
3
4 --custom data type
5
6 data Customer = Customer String String Double
7 deriving Show
8
9 mohitKumar :: Customer
10 mohitKumar = Customer "Mohit Kumar" "IITG" 12000.00
11
12 getBalance :: Customer -> Double
13 getBalance (Customer _ b) = b
14
15
```

Annotations in Notepad++:

- Blue arrows point from the "name" annotation to the string "Mohit Kumar".
- Blue arrows point from the "address" annotation to the string "IITG".
- Blue arrows point from the "balance" annotation to the number "12000.00".
- Blue arrows point from the "one double" annotation to the variable "b" in the `getBalance` function.
- Blue arrows point from the "two string" annotation to the two string fields in the `Customer` constructor.
- A blue arrow points from the "b represents balance" annotation to the variable "b" in the `getBalance` function.

Type Classes



The screenshot shows two windows side-by-side. On the left is the WinGHCi window, which displays the output of running a Haskell program. The output shows the loading of a module, the definition of two variables (*Main> isAnkitDarshit and *Main> ankitKumarData), and the definition of two other variables (*Main> darshitPatelData and *Main>). The right window is Notepad++ displaying the Haskell source code. The code defines a custom data type Employee with fields name, position, and idNum, and instances for Data.List and System.IO. It also includes code to check if two employees are equal and to show their details.

```

WinGHCi
File Edit Actions Tools Help
C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskelltutorial.hs
1 import Data.List
2 import System.IO
3
4 --custom data type
5
6 data Employee = Employee {name :: String,
7                           position :: String,
8                           idNum :: Int
9                           } deriving (Eq,Show)
10
11 ankitKumar = Employee{name="Ankit Kumar",position="Manager",idNum=1001}
12 darshitPatel = Employee{name="Darshit Patel",position="Sales",idNum=3001}
13
14 --check whether Ankit and Darshit are same or not
15 isAnkitDarshit = ankitKumar == darshitPatel
16
17 --shows the data of employee
18 ankitKumarData = show ankitKumar
19 darshitPatelData = show darshitPatel

```

able to show the employee details

able to check for the equality



Assignments

- Only THREE
 - Basic Haskell
 - Food bill generation system
 - Encryption system
- Will get more than ONE month to do (deadline: 30th November)
- Contact head TA (Subrata) for queries

END OF TUTORIAL

YOU MAY EXPLORE

<http://www.learnyouahaskell.com>

FOR MORE DETAIL
