

Functional Programming Concept with Haskell

Tutorial for
Programming Language Laboratory (CS 431)
July – November 2019

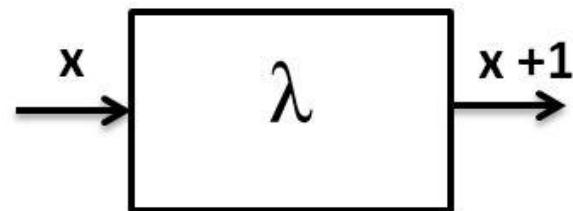


Functional Programming(FP)

- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism

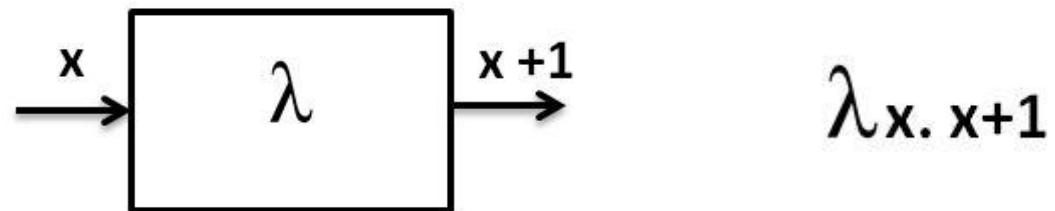
Functional Programming(FP)

- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism



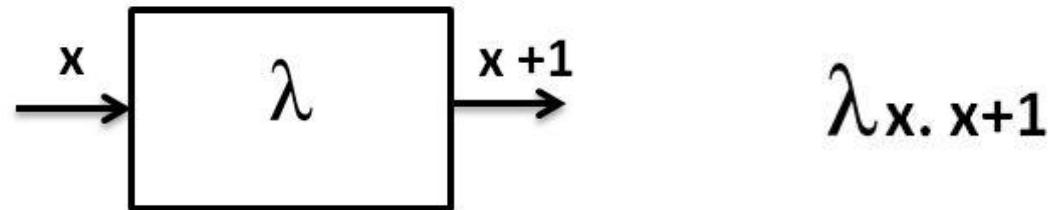
Functional Programming(FP)

- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism



Functional Programming(FP)

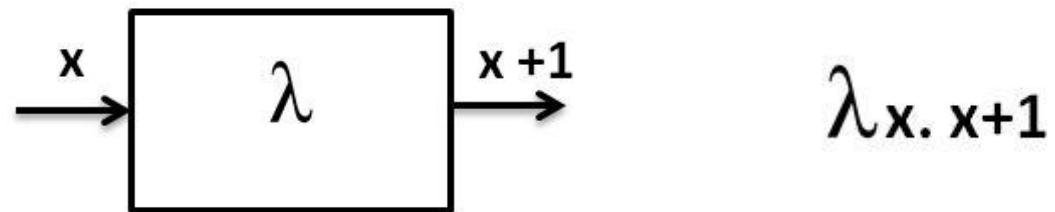
- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism



True:

Functional Programming(FP)

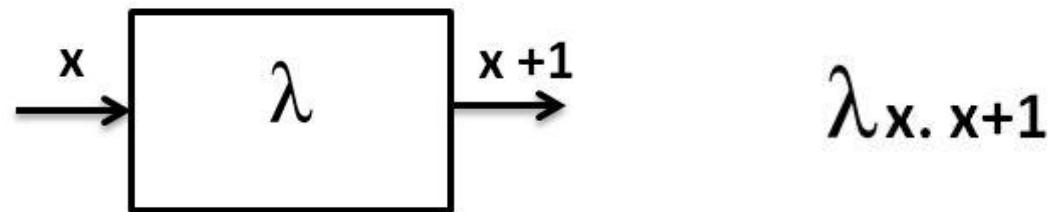
- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism



True: $\lambda x. \lambda y. x$

Functional Programming(FP)

- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism

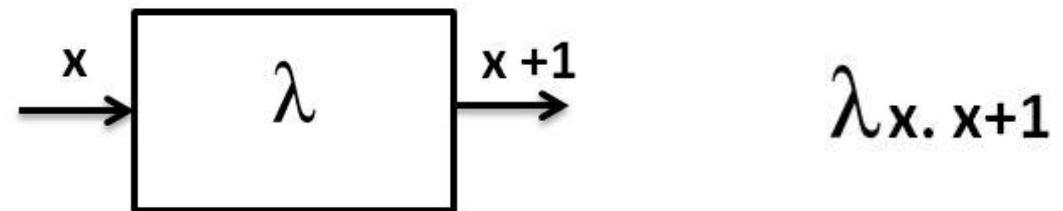


True: $\lambda x. \lambda y. x$

False:

Functional Programming(FP)

- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism



True: $\lambda x. \lambda y. x$ False: $\lambda x. \lambda y. y$

Functional Programming(FP)

- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism
- Languages that follow functional programming paradigm
 - Haskell
 - LISP
 - Python
 - Erlang
 - Racket
 - F#
 - Clojure
 - Scala

Functional Programming

- Key Idea - computation as ‘evaluation of mathematical functions’
 - Idea originated from Lambda Calculus formalism
- Languages that follow functional programming paradigm
 - Haskell
 - LISP
 - Python
 - Erlang
 - Racket
 - F#
 - Clojure
 - Scala

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”

Haskell - Features

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

Purely functional

- Every function in Haskell is a function in the mathematical sense (i.e., "pure")
 - The pure function returns the same output every time for the same input
 - In a pure functional language, you can't do anything that has a side effect

Purely functional

```
function impure(str: string){  
    str = str + "Post";  
    print(str);  
    return(str);  
}
```

State of function
gets changed

Ex. Impure function

```
function impure(str: string){  
    return(str + "Post");  
}
```

No change in state Immutable

Ex. Pure function

Statically Typed



- Every expression in Haskell has a type which compile time is determined at compile time
 - The compiler knows which piece of code is a number, which is a string and so on

```
haskell function = print "Hello Haskell learner"
```

variable

function

String

Compiler
automatically detects
the types

Statically Typed

- All the types composed together by function application have to match up. If they don't, the program will be rejected by the compiler

➤ Ex. `addMe :: Int -> Int -> Int` ← type signature or function declaration

➤ `addMe x y = x+y` ← function definition

```
*Main> addMe 4 5  
9
```

```
*Main> addMe 4 5.5
```

```
<interactive>:23:9: error:  
• No instance for (Fractional Int) arising from the literal '5.5'  
• In the second argument of 'addMe', namely '5.5'  
  In the expression: addMe 4 5.5  
  In an equation for 'it': it = addMe 4 5.5
```

Type Inference

- You don't have to explicitly label every piece of code because the type system can intelligently figure it out

Eg. If we write `a=5+4`

Haskell will automatically infer that `a` is a number

Lazy

- Nothing is evaluated unless it has to be

Eg. Function call : $f\ 5\ (29^{35792})$

Both the x and y values are evaluated and passed to function f

Non lazy languages like C or Java

Haskell pass the arguments value as it is without doing any actual computation of 29^{35792}

Haskell

Saves on CPU usage and user's time!

Concurrency

- Functional programming, by its nature (lack of side effect), is suitable for parallelism
- Concurrency in Haskell is mostly done with Haskell threads
- The Glasgow Haskell Compiler (GHC), comes with concurrency library containing a number of useful concurrency primitives and abstractions technique called Software Transactional Memory (STM)
- STM is an alternative to the lock based synchronization, whose basic objective is to evaluate a set of expression in isolated manner

Haskell - 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>atoparsec</u>	Fast parser
<u>monad-logger</u>	Logging	<u>persistent</u>	Database ORM
<u>template-haskell</u>	Meta-programming	<u>tar</u>	Tar archives

Haskell - Application





Haskell - Application

facebook

Haskell - Application

- **facebook** Anti-spam programs

Haskell - Application

- **facebook** Anti-spam programs



Haskell - Application

-  anti-spam programs
-  a window manager for the X Window System
xmonad

Haskell - Application

-  anti-spam programs
-  a window manager for the X Window System
xmonad



Haskell - Application

-  anti-spam programs
-  a window manager for the X Window System
xmonad
-  **darcs** revision control system

Some other FP Applications



Some other FP Applications



Some other FP Applications

-  

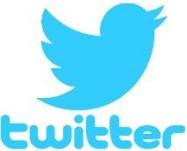
Some other FP Applications

-  
- 

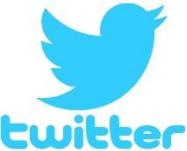
Some other FP Applications

-  
-  

Some other FP Applications

-  
-  
- 

Some other FP Applications

-   Scala
-   Erlang
-   Lisp

Haskell

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

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>

haskell-tutorial - Notepad

File Edit Format View Help

```
main = do
    putStrLn "This Works!"
```

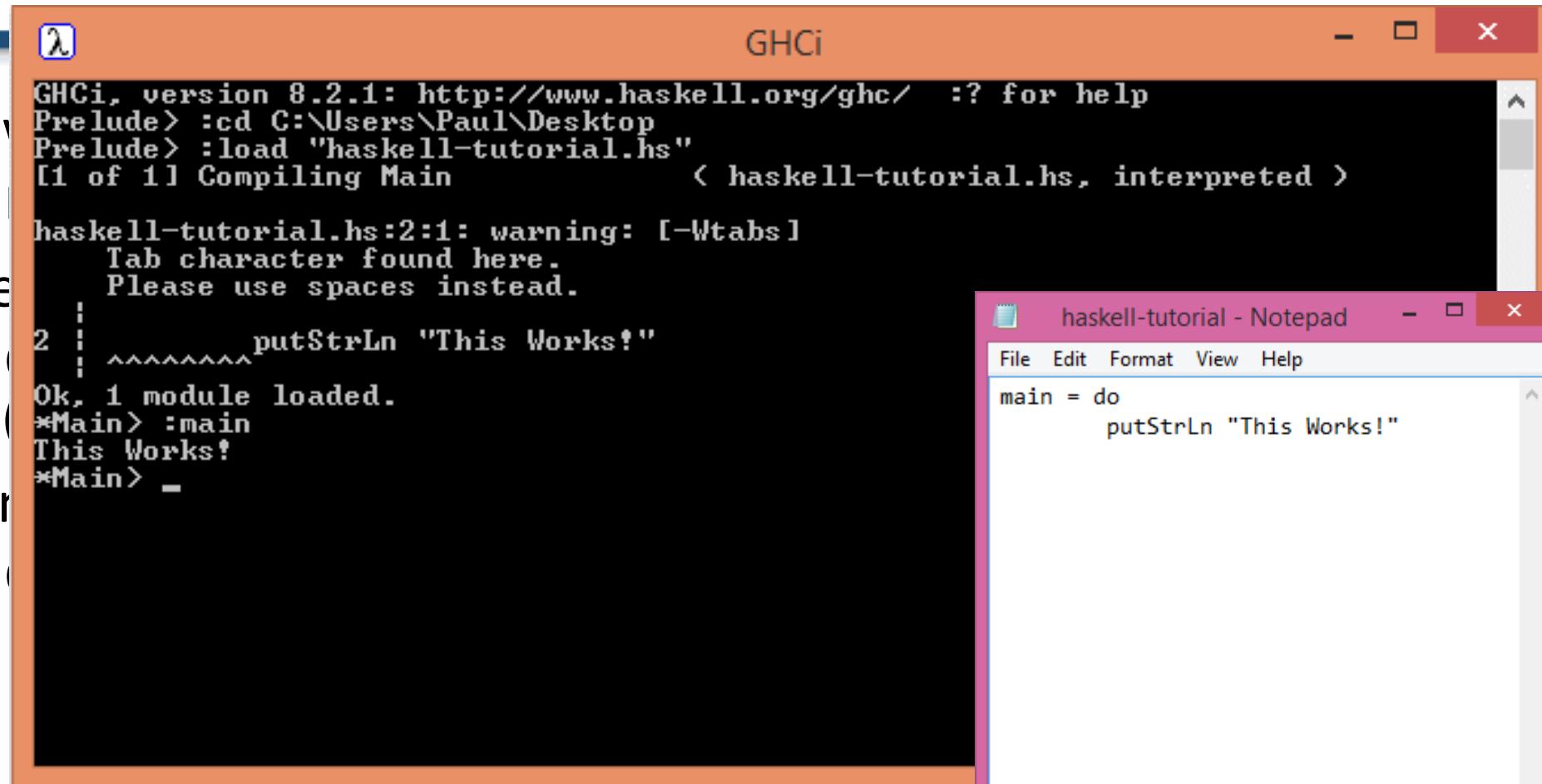
# Run your First Haskell Program

---

- 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

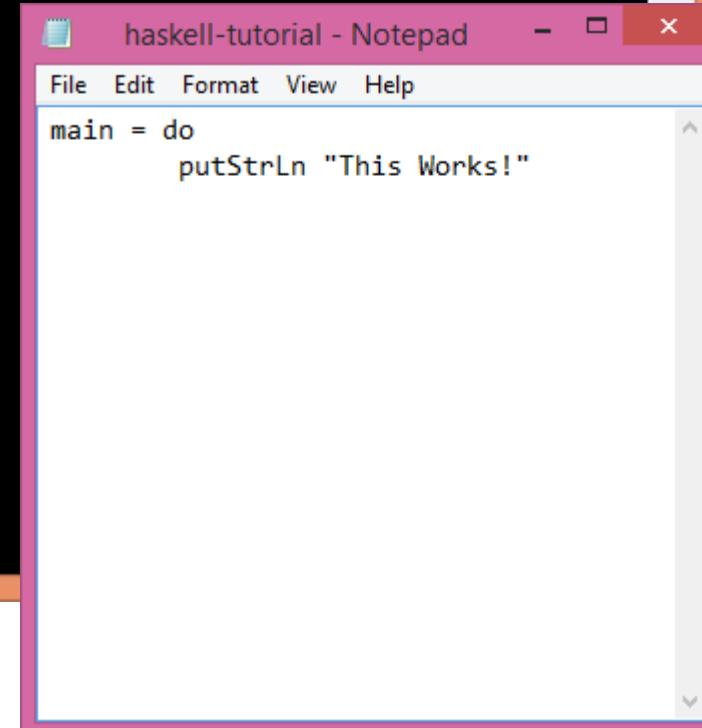
- Doing
- File
- Configuration
- Compiler
- 



GHCI

```
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> _
```

- Run the function you want



haskell-tutorial - Notepad

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

ion  
p)

# Few things you may keep in mind

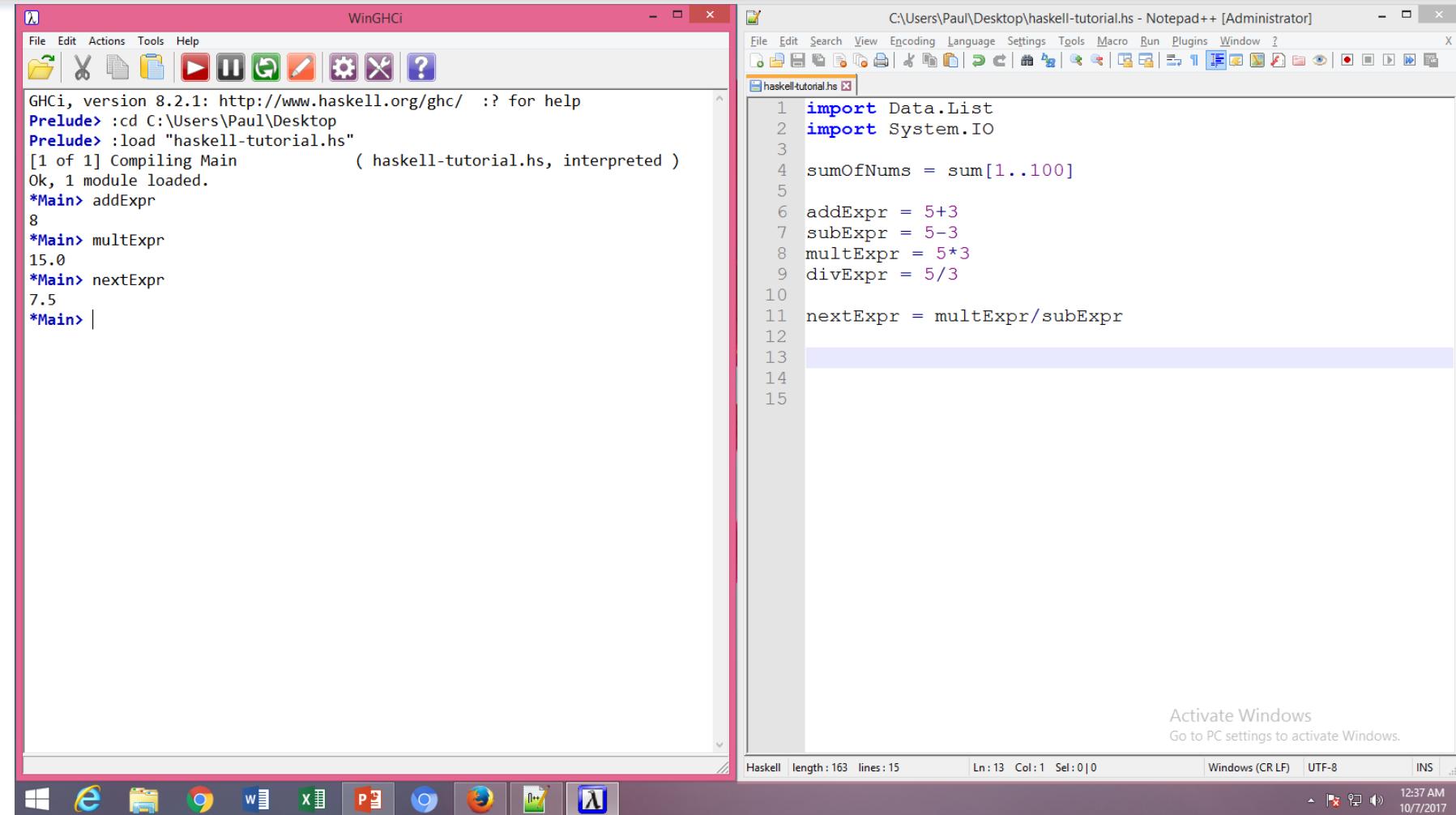
---

- 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: Float, Double, Bool, Char, Tuple -- will be discussing with example
    - permanent3 :: Int  
permanent3 = 3
- permanent value for a variable  
--Never Change

# Expressions



The image shows a Windows desktop environment with two open windows. On the left is the WinGHCi window, which displays a Haskell interactive session. The session starts with the GHCi version (8.2.1) and help information. It then changes directory to the user's desktop and loads a file named "haskell-tutorial.hs". The session then defines several variables: addExpr (5+3), subExpr (5-3), multExpr (5\*3), and divExpr (5/3). Finally, it calculates nextExpr as multExpr divided by subExpr, resulting in 7.5. The right window is Notepad++, showing the source code for "haskell-tutorial.hs". The code imports Data.List and System.IO, defines sumOfNums as the sum of numbers from 1 to 100, and then defines the four expressions: addExpr, subExpr, multExpr, and divExpr. The Notepad++ status bar at the bottom indicates the file is in Haskell mode, has a length of 163 and 15 lines, and is in Windows (CR LF) encoding.

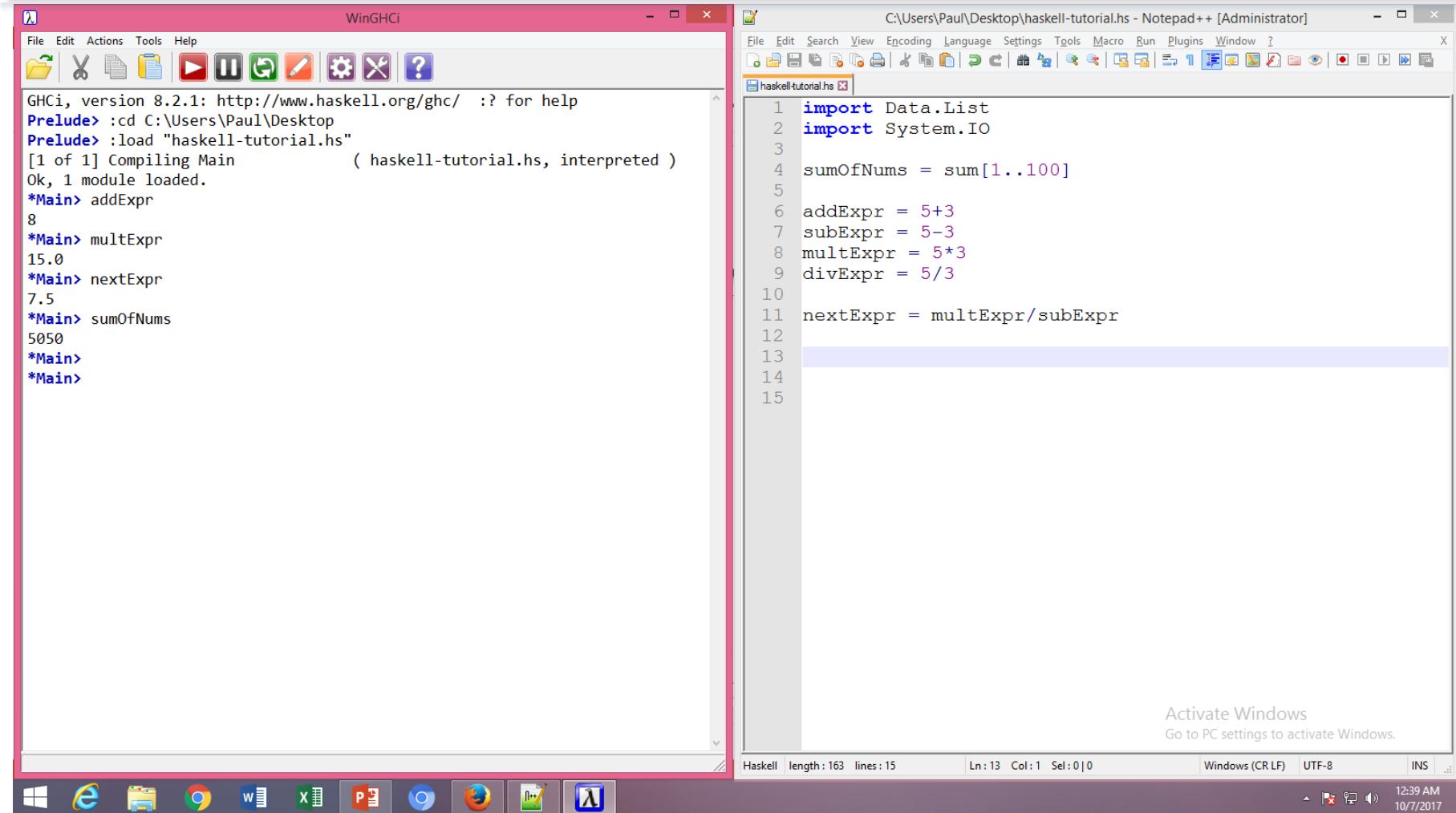
```
WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/  ?: for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main             ( haskell-tutorial.hs, interpreted )
Ok, 1 module loaded.
*Main> addExpr
8
*Main> multExpr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

Activate Windows  
Go to PC settings to activate Windows.

Haskell length : 163 lines : 15 Ln : 13 Col : 1 Sel : 0|0 Windows (CR LF) UTF-8 INS 12:37 AM 10/7/2017

# Expressions



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

- WinGHCi**: A terminal window for the Haskell Compiler Interactive Environment (GHCi). 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.hs, interpreted )
Ok, 1 module loaded.
*Main> addExpr
8
*Main> multExpr
15.0
*Main> nextExpr
7.5
*Main> sumOfNums
5050
*Main>
*Main>
```

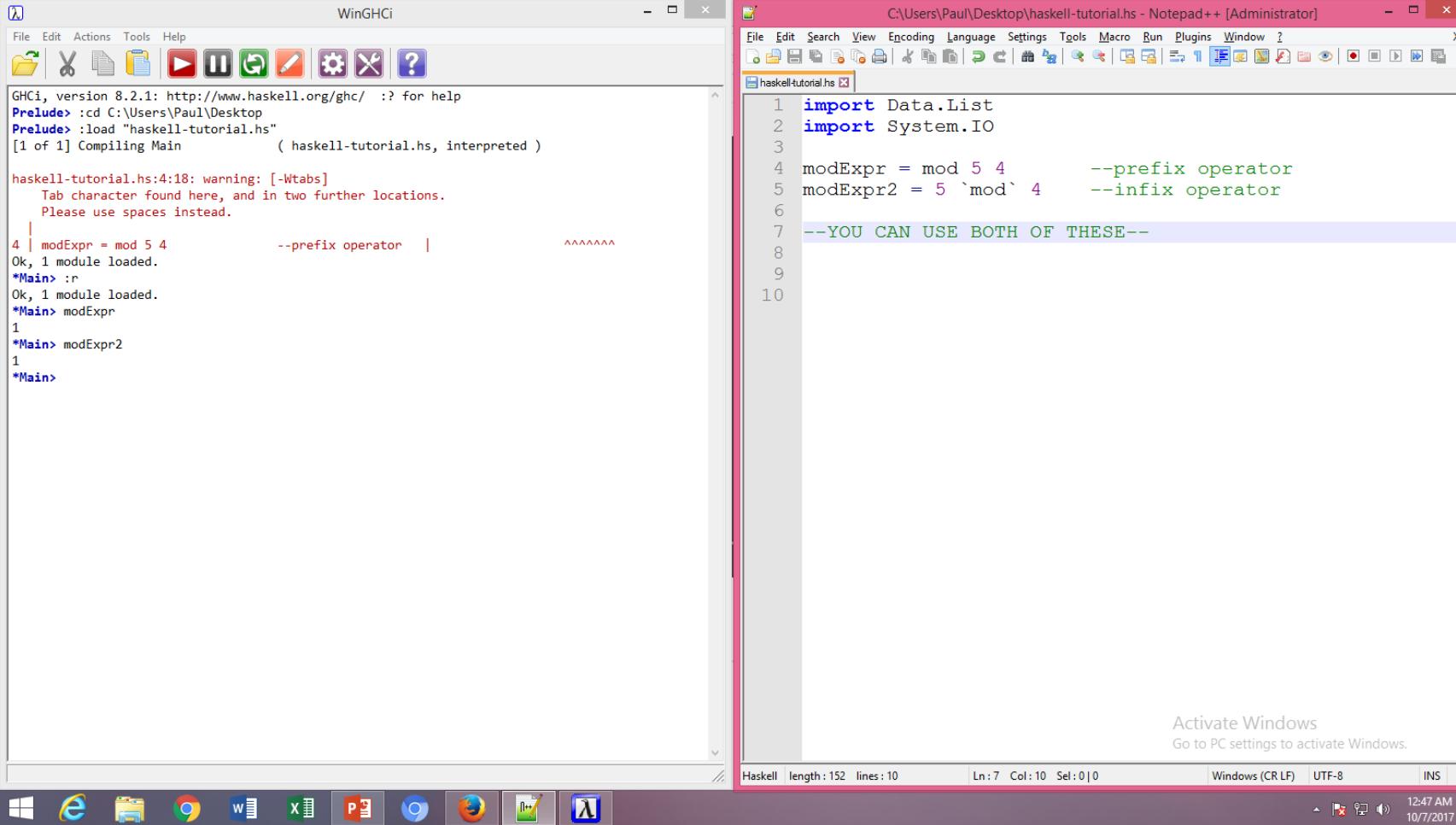
- C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]**: A code editor window showing the Haskell source code for `haskell-tutorial.hs`. The code defines several functions:

```

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
```

The Notepad++ status bar at the bottom indicates the file is in Haskell mode, with statistics: length: 163, lines: 15, Ln: 13, Col: 1, Sel: 0|0, and encoding: Windows (CR LF), UTF-8, INS.

# Infix and Prefix Operator



The image shows two windows side-by-side. On the left is the WinGHCi window, which displays a Haskell interpreter session. The session starts with GHCi version 8.2.1, then changes the directory to the desktop, loads the file 'haskell-tutorial.hs', and enters the main module. It then shows a warning about tabs and proceeds to define two functions: 'modExpr' using a prefix operator and 'modExpr2' using an infix operator. Both definitions are followed by a comment indicating they can be used interchangeably.

```
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 )

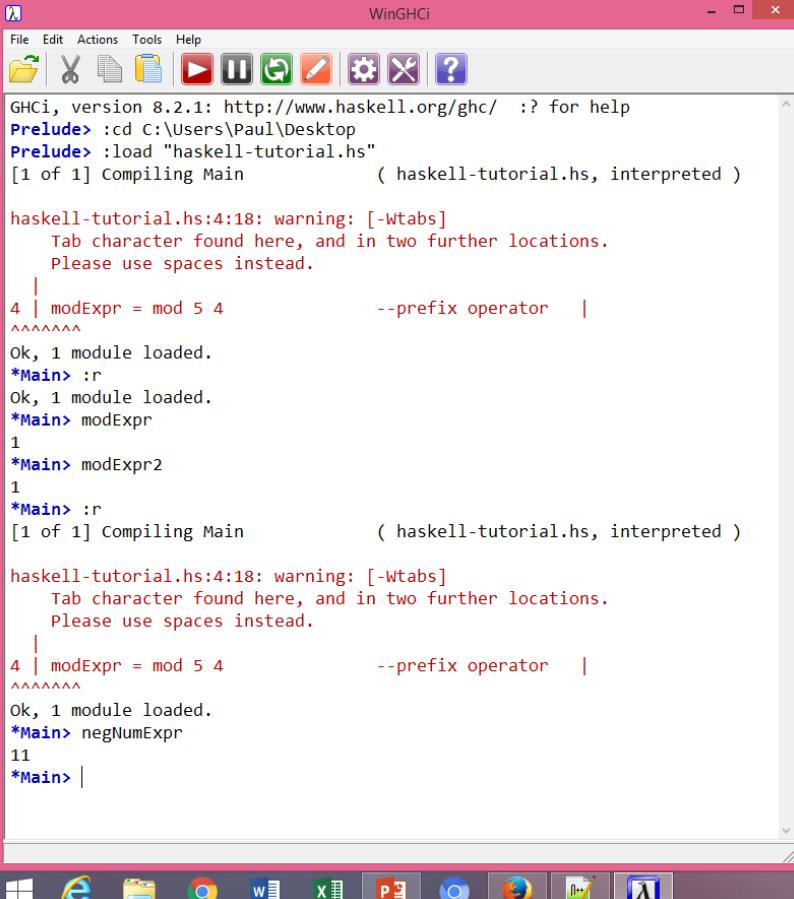
haskell-tutorial.hs:4:18: warning: [-Wtabs]
  Tab character found here, and in two further locations.
  Please use spaces instead.

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

The right window is Notepad++ showing the source code 'haskell-tutorial.hs'. It contains the same code as the GHCi session, with syntax highlighting for keywords like 'import', operators like 'mod', and comments. A purple horizontal bar highlights the text '---YOU CAN USE BOTH OF THESE---'.

```
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



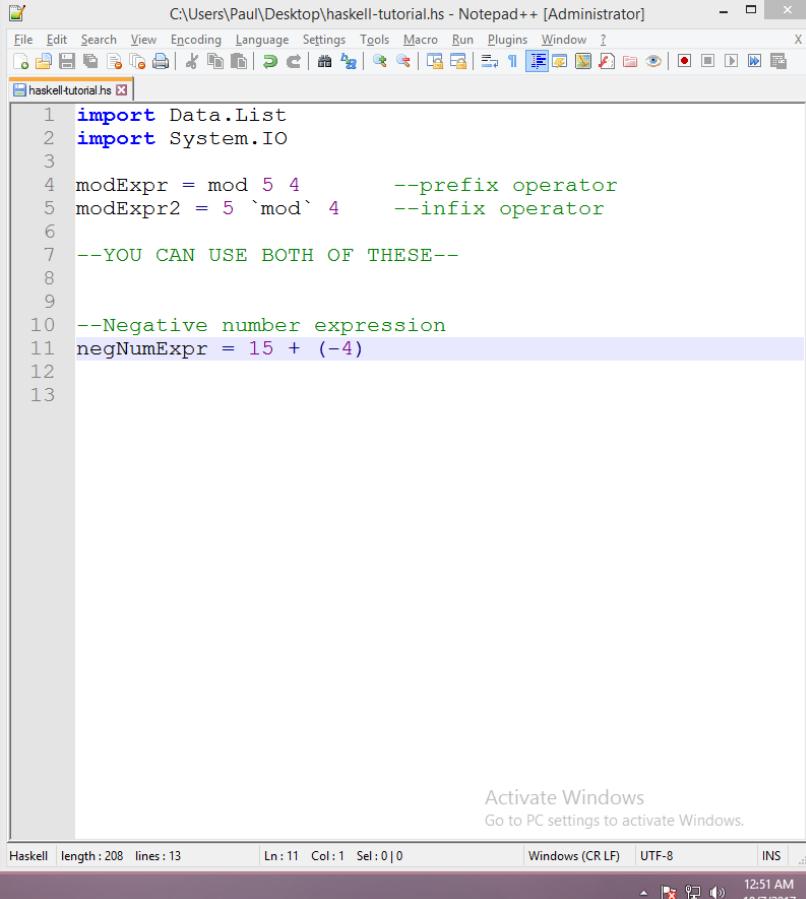
```

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
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

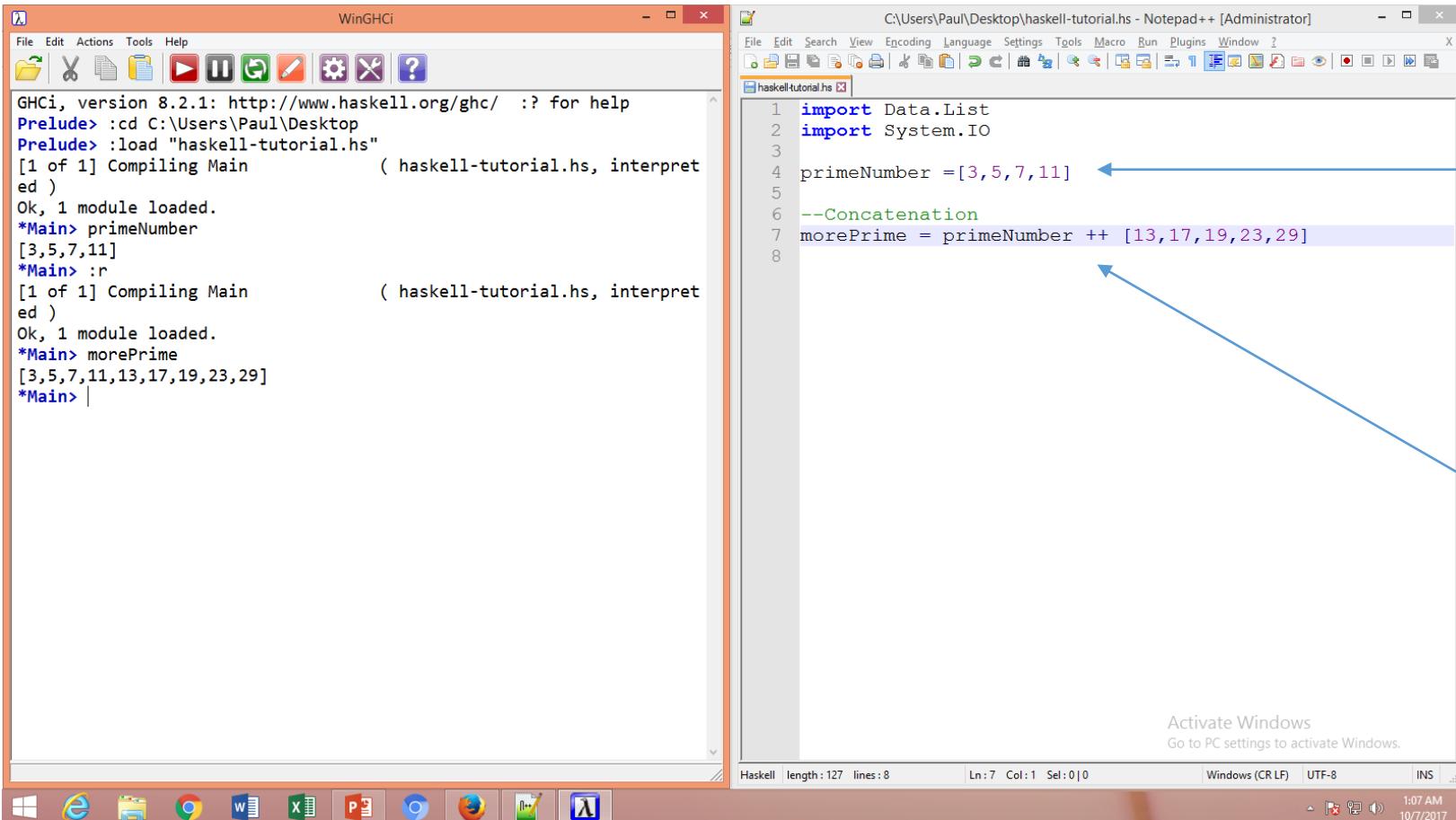
- `piVal = pi`
- `ePow9 = exp 9`
- `logOf9 = log 9`
- `Squared9 = 9 ** 2`
- `truncateVal = truncate 9.999`
- `roundVal = round 9.999`
- `ceilingVal = ceiling 9.999`
- `floorVal = floor 9.999`
- Also
  - `sin, cos, tan, asin, acos, atan, signh, cosh, tanh, asinh, acosh, atanh`

# 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, signh, 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 interactive interpreter. It displays the following session:

```

WinGHCi
File Edit Actions Tools Help
[1] Prelude> :cd C:\Users\Paul\Desktop
[2] Prelude> :load "haskell-tutorial.hs"
[3] [1 of 1] Compiling Main             ( haskell-tutorial.hs, interpreted )
[4] Ok, 1 module loaded.
[5] *Main> primeNumber
[6] [3,5,7,11]
[7] *Main> :r
[8] [1 of 1] Compiling Main             ( haskell-tutorial.hs, interpreted )
[9] Ok, 1 module loaded.
[10] *Main> morePrime
[11] [3,5,7,11,13,17,19,23,29]
[12] *Main>

```

On the right is the Notepad++ window, showing 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

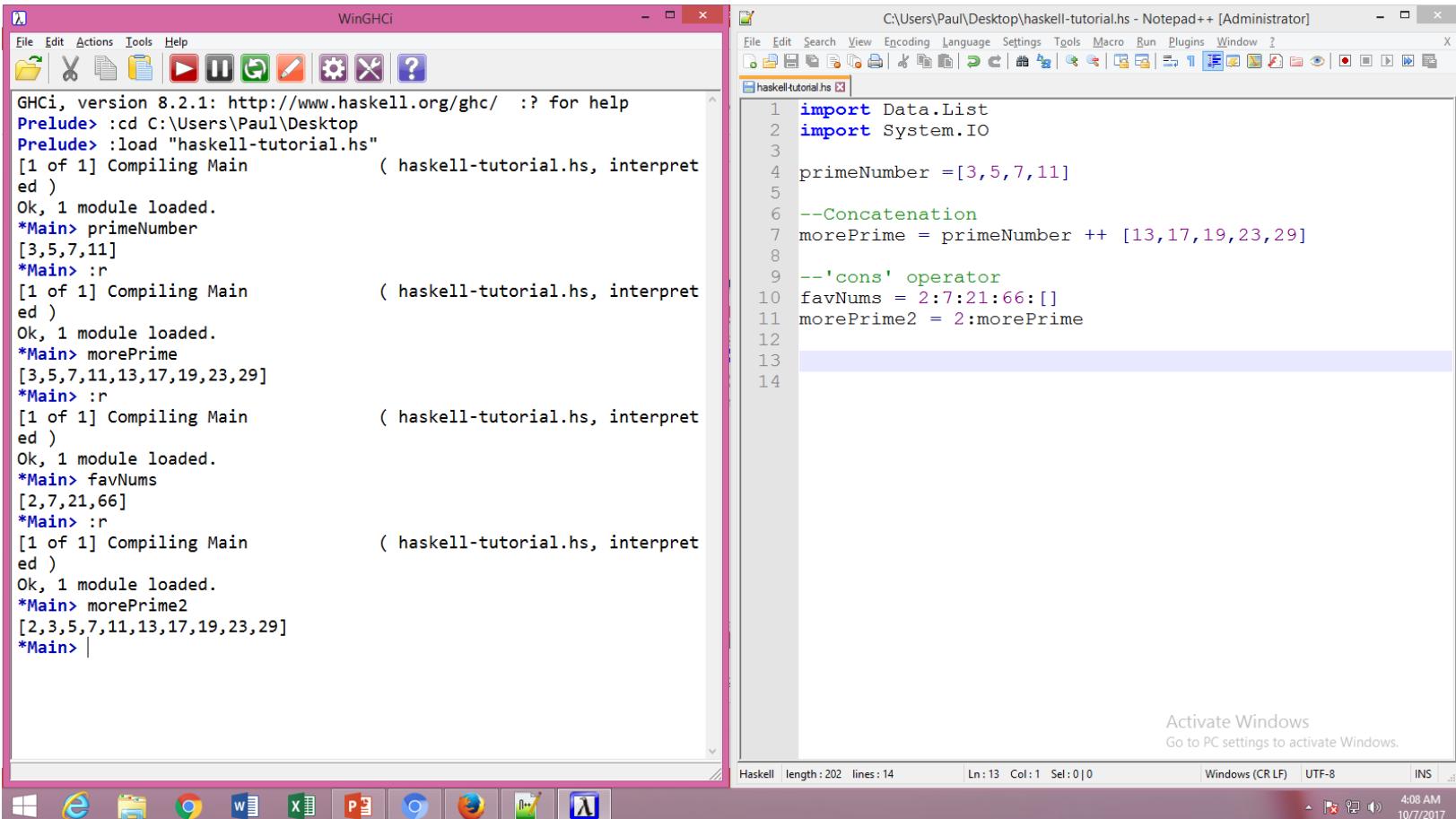
```

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]`.

Define a list

Concatenation  
of two lists

# List – ‘cons’ operator



**WinGHCi**

```

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> 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> :r
[1 of 1] Compiling Main           ( haskell-tutorial.hs, interpreted )
Ok, 1 module loaded.
*Main> favNums
[2,7,21,66]
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutorial.hs, interpreted )
Ok, 1 module loaded.
*Main> morePrime2
[2,3,5,7,11,13,17,19,23,29]
*Main>

```

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

```

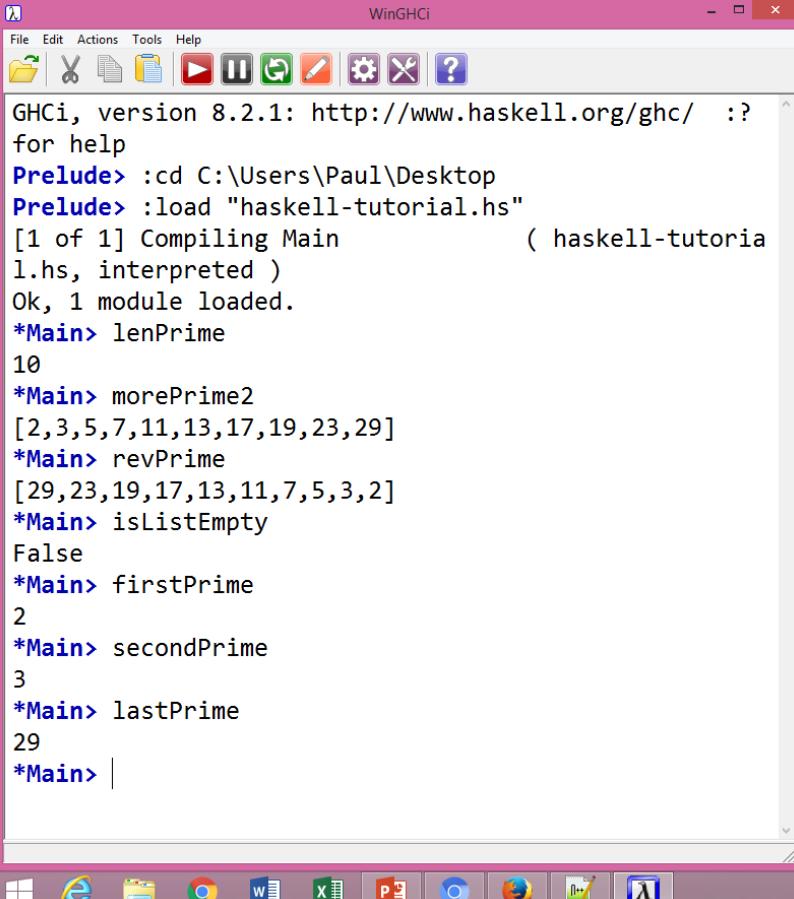
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13
14

```

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

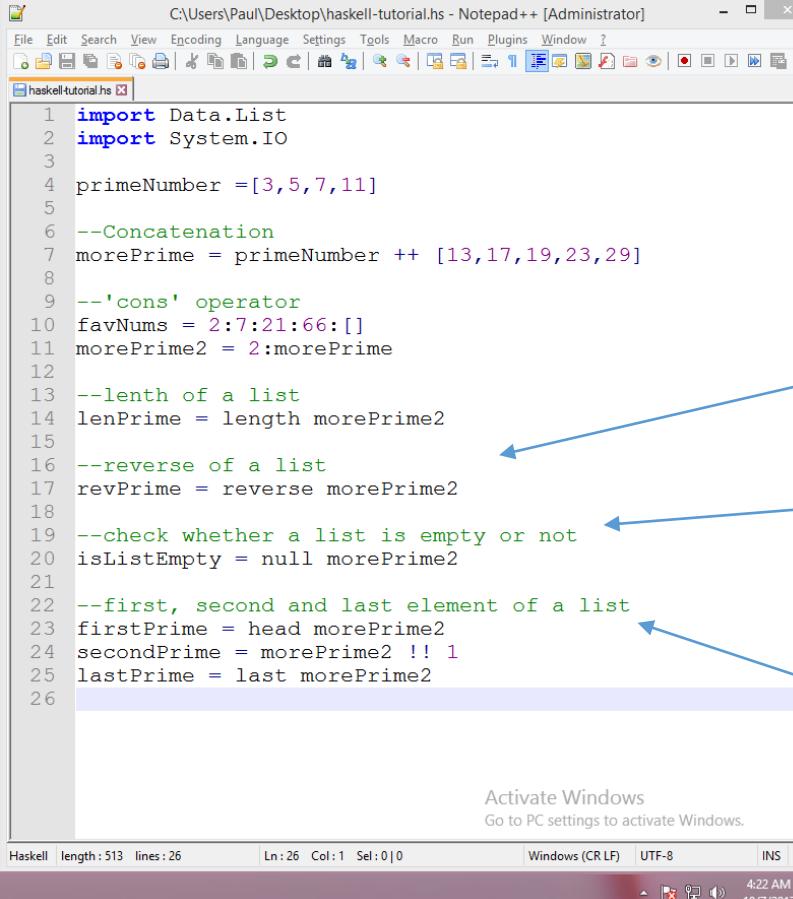
# More Operations on List



```

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> 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>

```



```

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 --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

```

Activate Windows  
Go to PC settings to activate Windows.

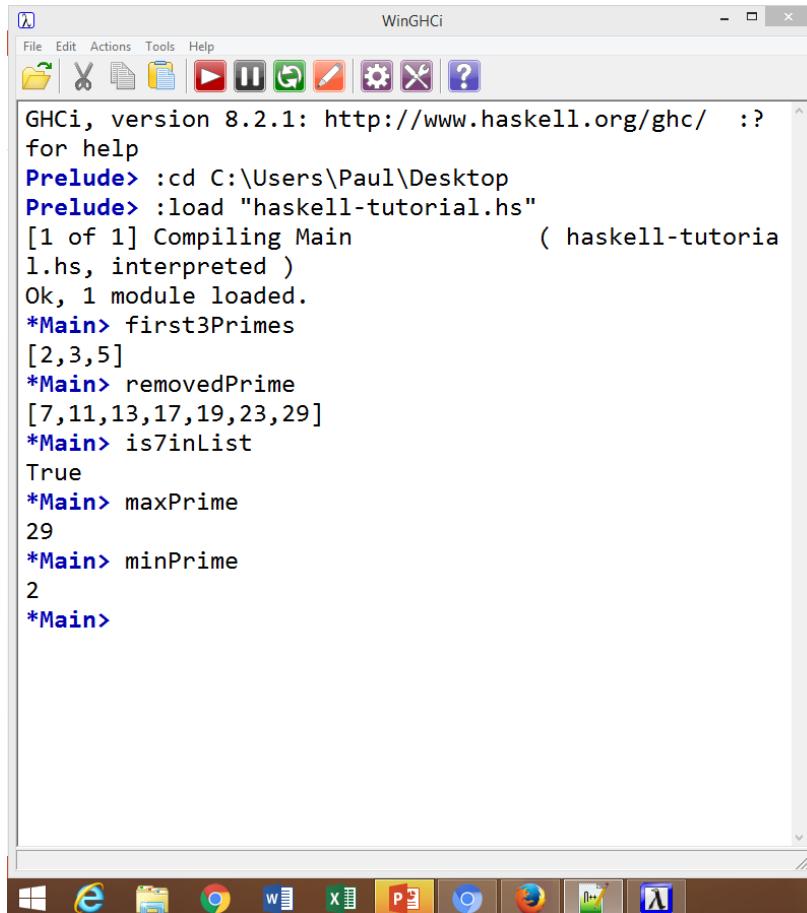
Haskell length : 513 lines : 26 Ln : 26 Col : 1 Sel : 0 | 0 Windows (CR LF) UTF-8 INS 4:22 AM 10/7/2017

reverse

List empty?

particular element

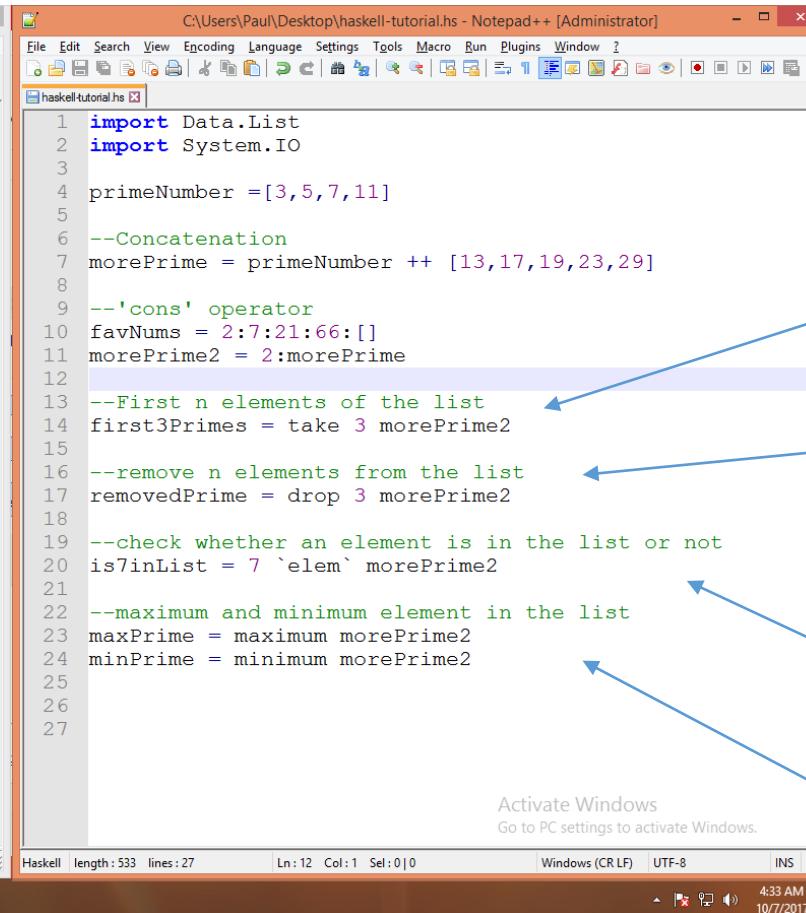
# More Operations on List



```

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>

```



```

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

```

Activate Windows  
Go to PC settings to activate Windows.

Windows (CR LF) UTF-8 INS

4:33 AM 10/7/2017

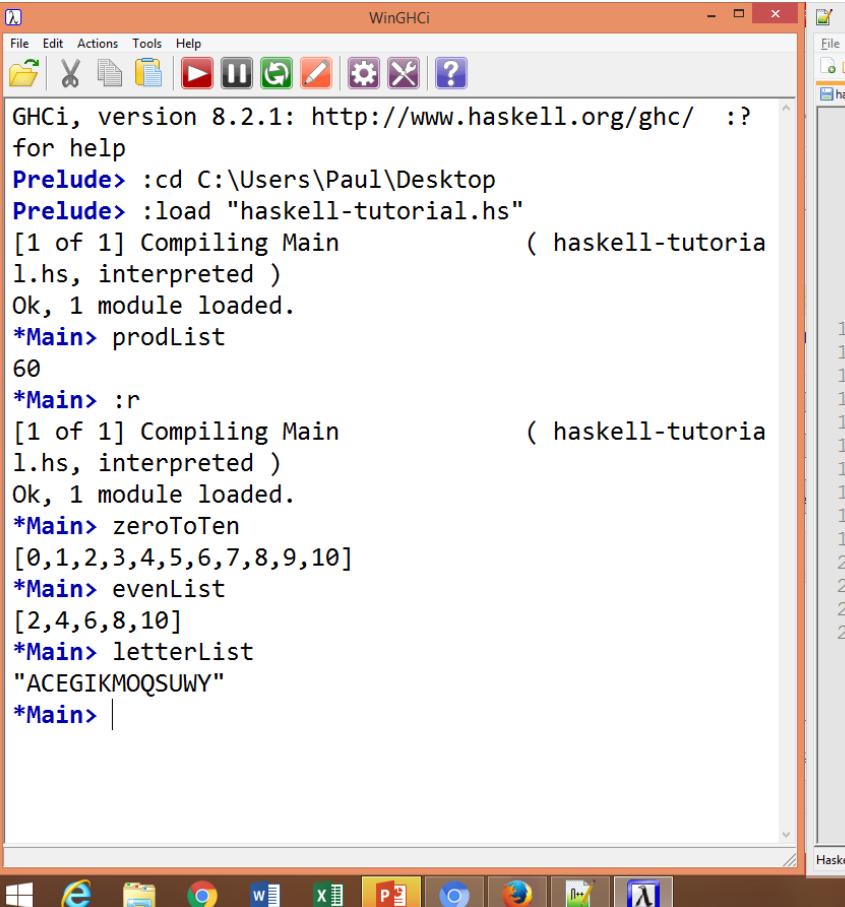
first n element

removing first n element

finding an element

max and min

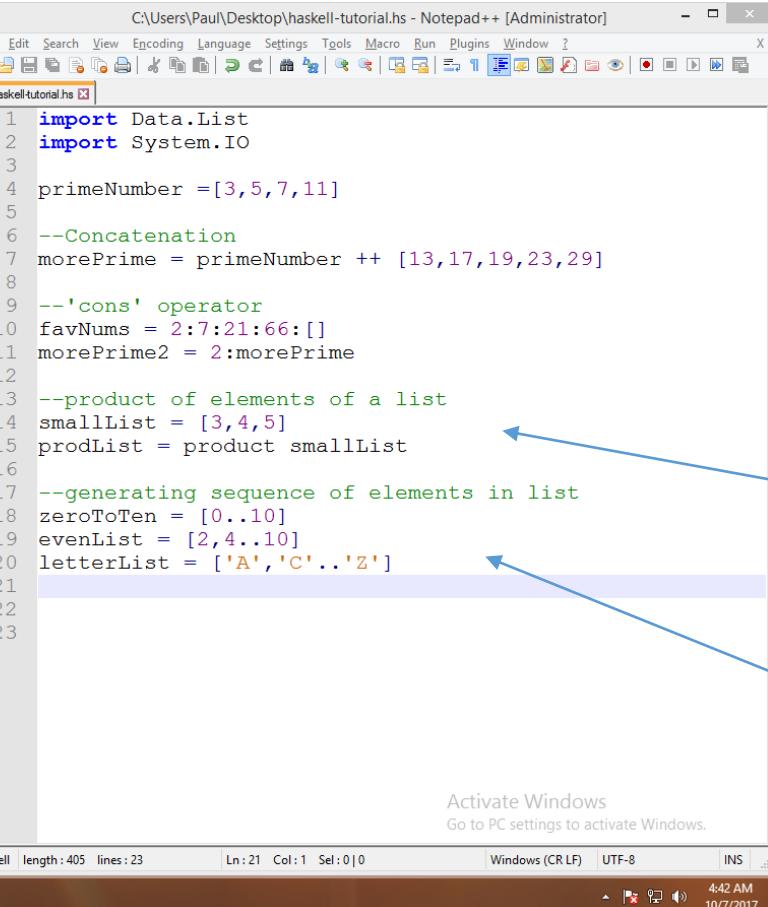
# More Operations on List



```

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>

```



```

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 --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']

Activate Windows
Go to PC settings to activate Windows.

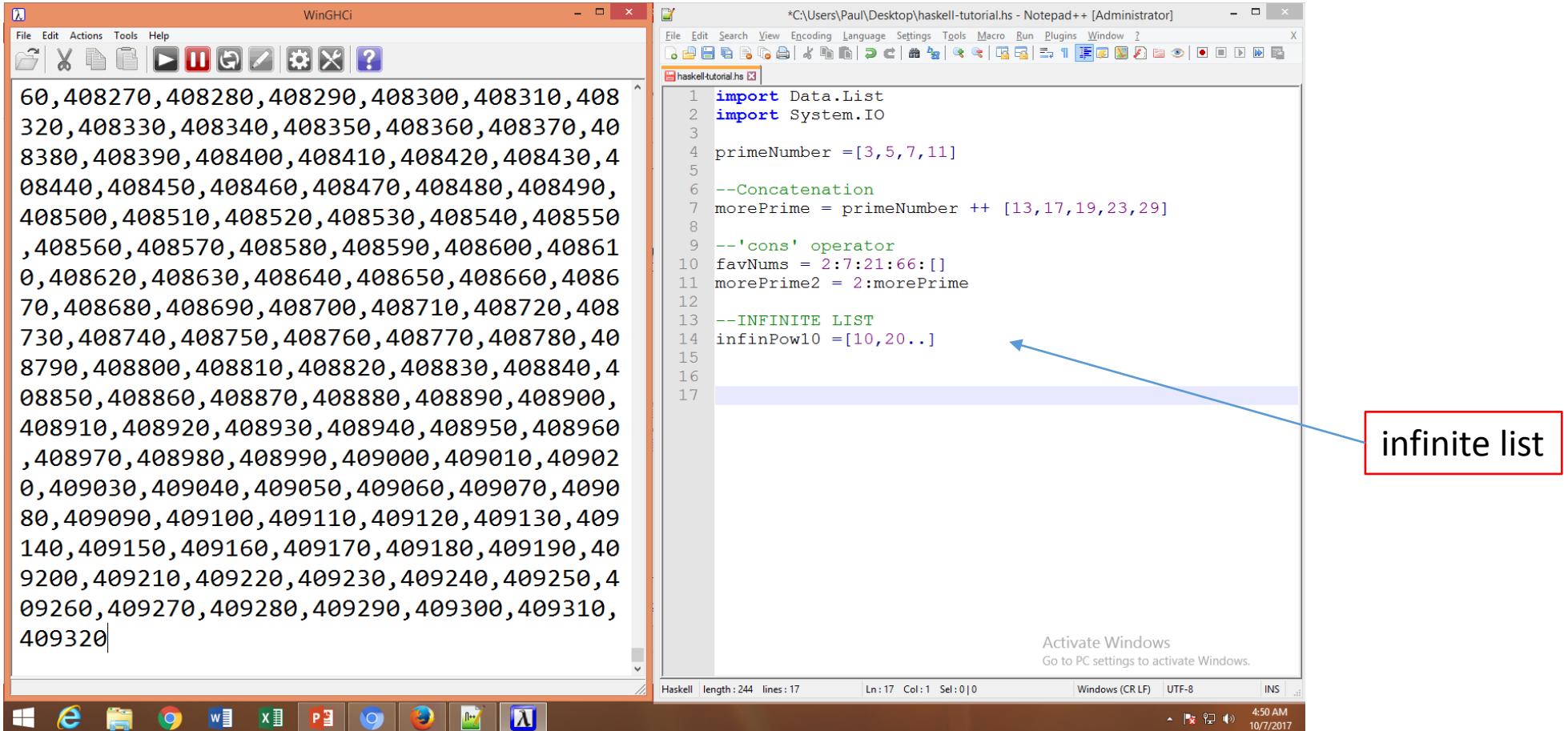
Haskell length: 405 lines: 23 Ln:21 Col:1 Sel:0|0 Windows (CR LF) UTF-8 INS ↻ 4:42 AM 10/7/2017

```

**product**

**sequence**

# More Operations on List



The image shows two windows side-by-side. On the left is the WinGHCi interface, which displays a long list of integers starting from 60 and ending at 409320. On the right is a Notepad++ window titled 'haskell-tutorial.hs' containing Haskell code. The code imports Data.List and System.IO, defines primeNumber as [3, 5, 7, 11], and uses concatenation to create morePrime. It also demonstrates the 'cons' operator to create favNums and morePrime2. Finally, it creates an infinite list 'infinPow10' starting from 10. A red box highlights the 'infinPow10' line, and a blue arrow points from this box to the word 'infinite list' located in the bottom right corner of the slide.

```

WinGHCi
File Edit Actions Tools Help
60,408270,408280,408290,408300,408310,408
320,408330,408340,408350,408360,408370,408
8380,408390,408400,408410,408420,408430,408
440,408450,408460,408470,408480,408490,
408500,408510,408520,408530,408540,408550
,408560,408570,408580,408590,408600,40861
0,408620,408630,408640,408650,408660,4086
70,408680,408690,408700,408710,408720,408
730,408740,408750,408760,408770,408780,40
8790,408800,408810,408820,408830,408840,4
08850,408860,408870,408880,408890,408900,
408910,408920,408930,408940,408950,408960
,408970,408980,408990,409000,409010,40902
0,409030,409040,409050,409060,409070,4090
80,409090,409100,409110,409120,409130,409
140,409150,409160,409170,409180,409190,40
9200,409210,409220,409230,409240,409250,4
09260,409270,409280,409290,409300,409310,
409320

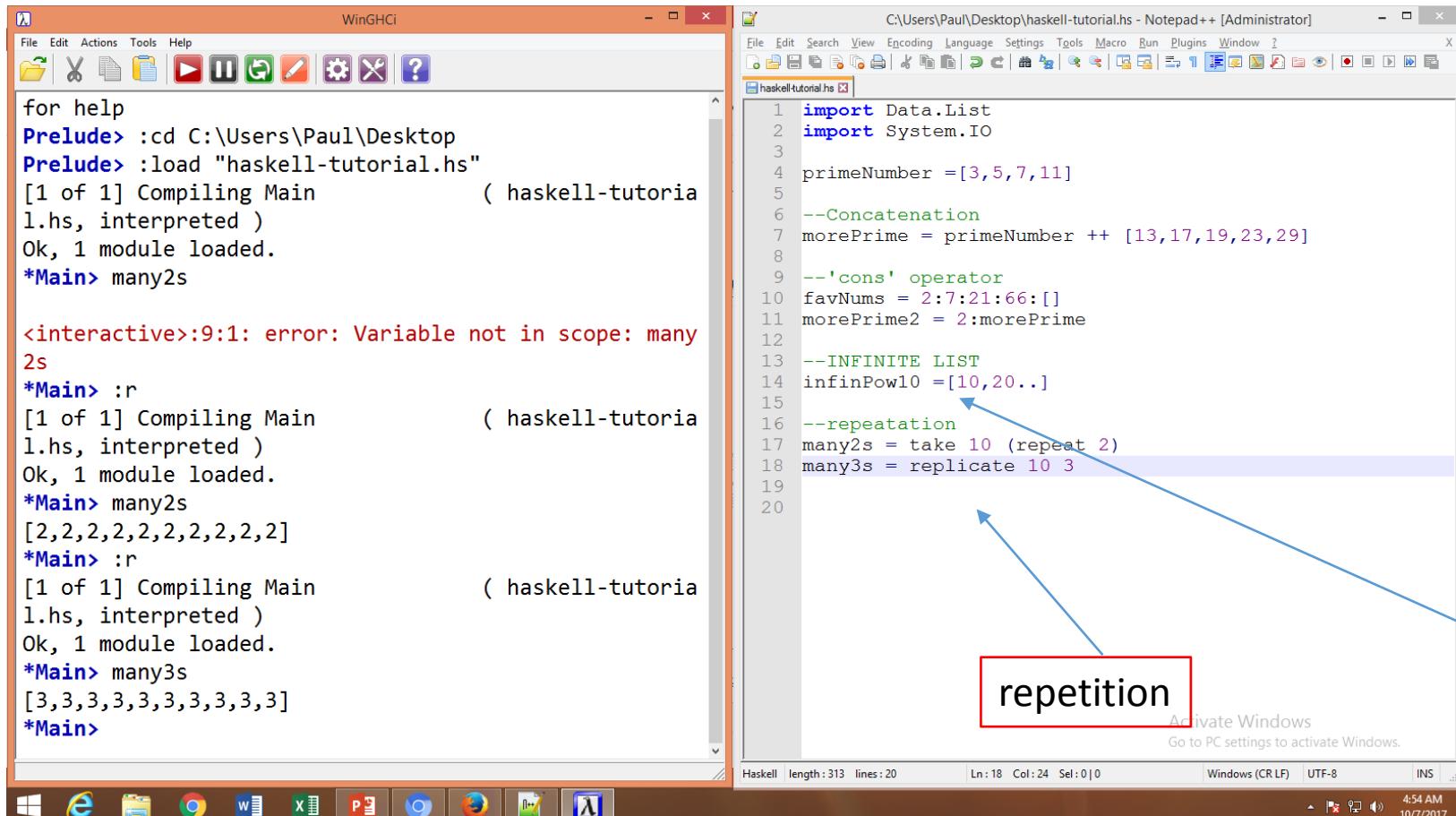
*haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --INFINITE LIST
14 infinPow10 =[10,20..]

Activate Windows
Go to PC settings to activate Windows.

Haskell length : 244 lines : 17 Ln : 17 Col : 1 Sel : 0 | 0 Windows (CR LF) UTF-8 INS
4:50 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 the following session:

```

WinGHCi
File Edit Actions Tools Help
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> many2s

<interactive>:9:1: error: Variable not in scope: many
2s
*Main> :r
[1 of 1] Compiling Main      ( haskell-tutoria
1.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
1.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 contents of the file `haskell-tutorial.hs`:

```

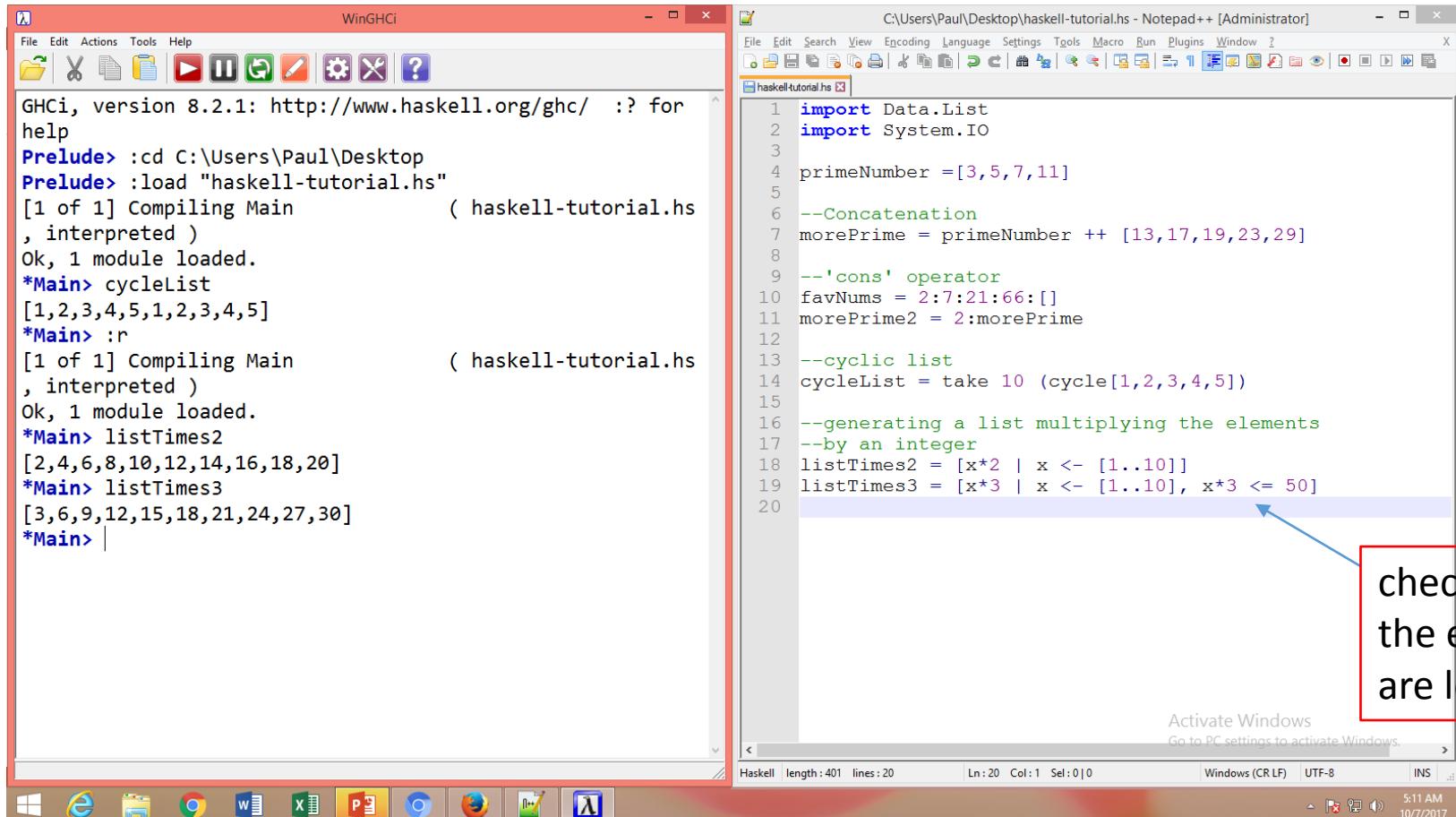
C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --INFINITE LIST
14 infinPow10 =[10,20..]
15
16 --repeation
17 many2s = take 10 (repeat 2)
18 many3s = replicate 10 3
19
20

```

A red box highlights the word `repetition` in the code. Three blue arrows point from the text "repetition" to the code lines `many2s = take 10 (repeat 2)`, `many3s = replicate 10 3`, and `infinPow10 =[10,20..]`.

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 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
[...]
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> cycleList
[1,2,3,4,5,1,2,3,4,5]
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutorial.hs
, interpreted )
Ok, 1 module loaded.
*Main> listTimes2
[2,4,6,8,10,12,14,16,18,20]
*Main> listTimes3
[3,6,9,12,15,18,21,24,27,30]
*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 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 --cyclic list
14 cycleList = take 10 (cycle[1,2,3,4,5])
15
16 --generating a list multiplying the elements
17 --by an integer
18 listTimes2 = [x*2 | x <- [1..10]]
19 listTimes3 = [x*3 | x <- [1..10], x*3 <= 50]
20

```

A red box with a blue arrow points from the text "check whether the element generated are less than 50 or not" to the line `listTimes3 = [x*3 | x <- [1..10], x*3 <= 50]`.

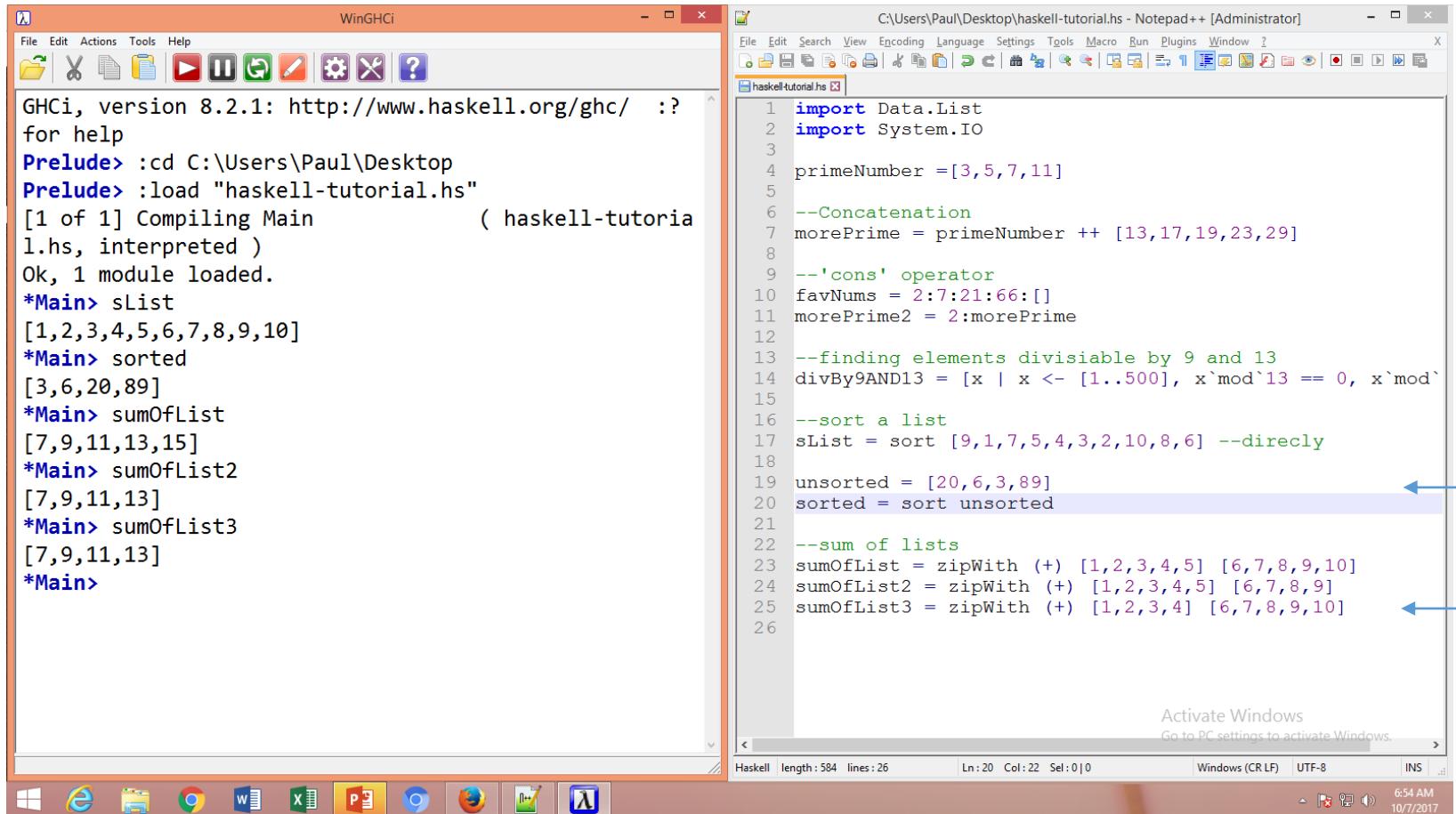
Activate Windows  
Go to PC settings to activate Windows.

Haskell length:401 lines:20 Ln:20 Col:1 Sel:0|0 Windows (CR LF) UTF-8 INS ...

5:11 AM 10/7/2017

check whether  
the element generated  
are less than 50 or not

# More Operations on List



The image shows two windows demonstrating Haskell list operations:

**WinGHCi Window (Left):**

```

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> 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>

```

**Notepad++ Window (Right):**

```

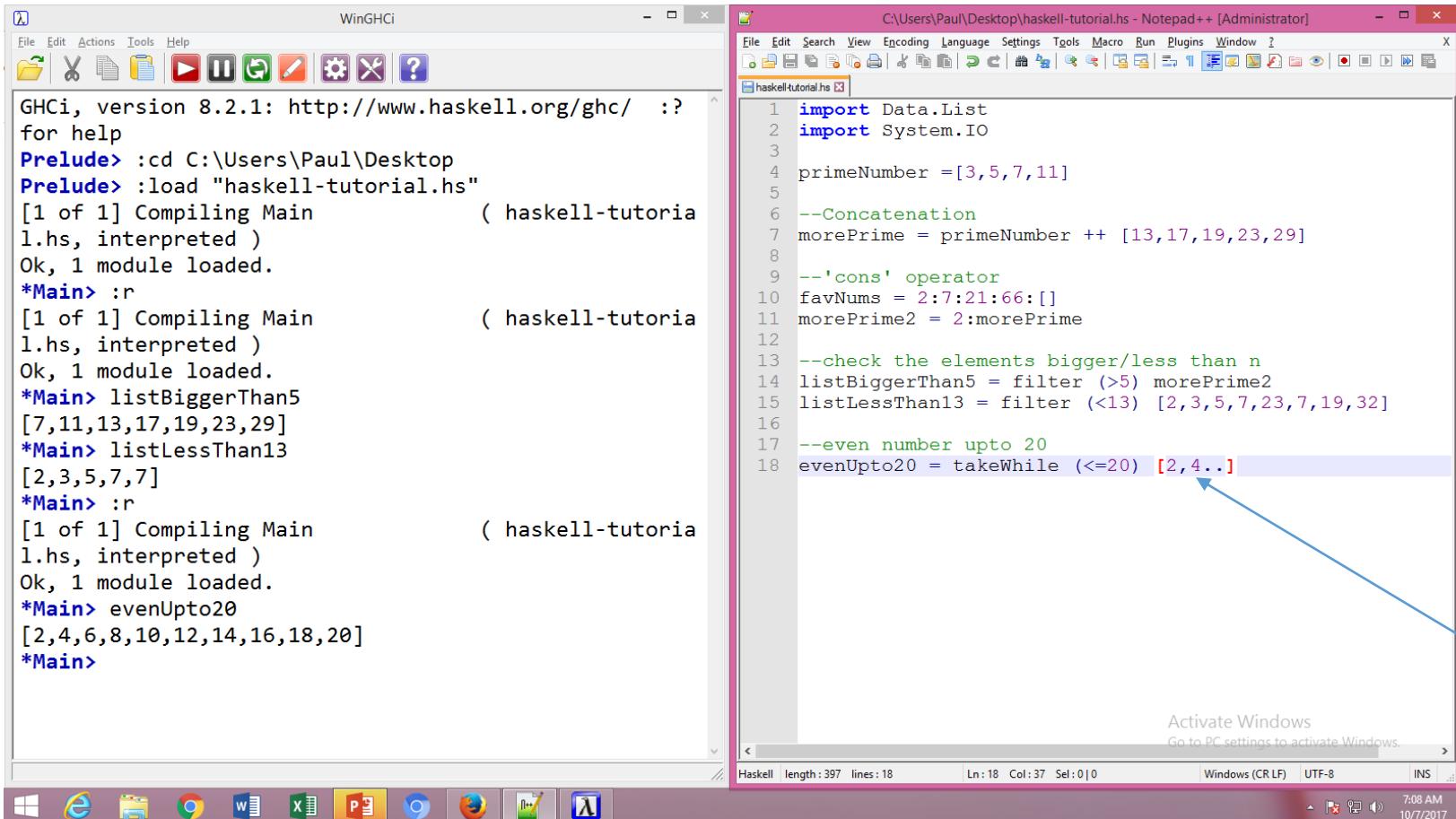
C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs x
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --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]

```

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

- A red box labeled "sorting" points to the line `sList = sort [9,1,7,5,4,3,2,10,8,6]`.
- A red box labeled "summation" points to the line `sumOfList = zipWith (+) [1,2,3,4,5] [6,7,8,9,10]`.

# More Operations on List



The image shows two windows side-by-side. On the left is the WinGHCi window, which displays a Haskell REPL session. On the right is the Notepad++ window containing Haskell code.

**WinGHCi Window:**

```

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> :r
[1 of 1] Compiling Main           ( haskell-tutoria
1.hs, interpreted )
Ok, 1 module loaded.
*Main> listBiggerThan5
[7,11,13,17,19,23,29]
*Main> listLessThan13
[2,3,5,7,7]
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutoria
1.hs, interpreted )
Ok, 1 module loaded.
*Main> evenUpto20
[2,4,6,8,10,12,14,16,18,20]
*Main>

```

**Notepad++ Window:**

```

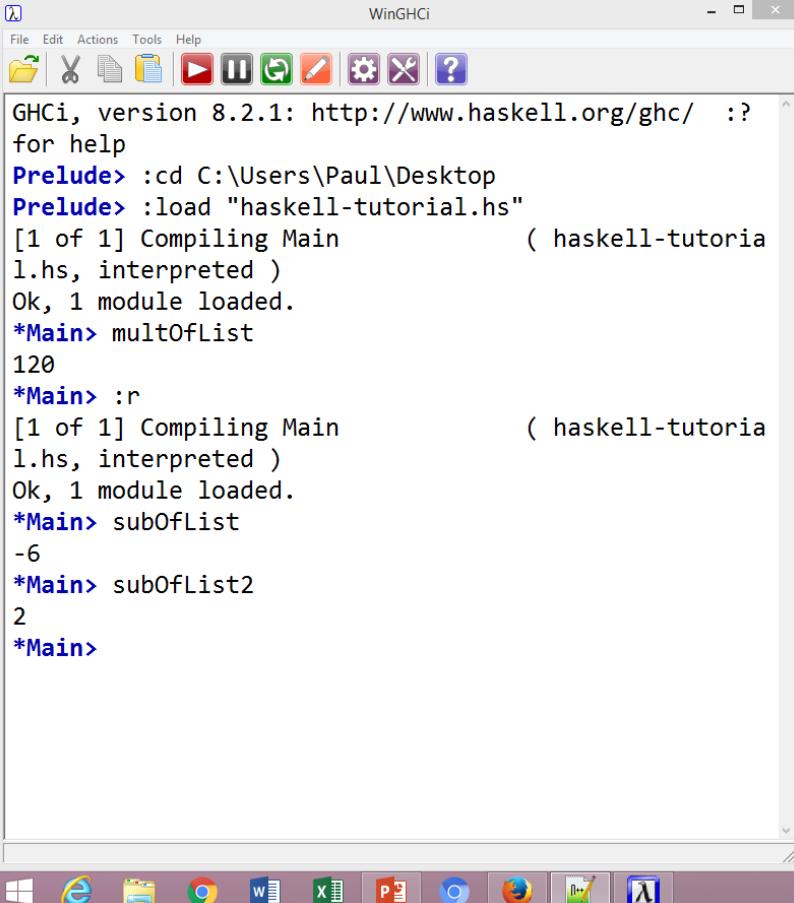
C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs x
1 import Data.List
2 import System.IO
3
4 primeNumber =[3,5,7,11]
5
6 --Concatenation
7 morePrime = primeNumber ++ [13,17,19,23,29]
8
9 --'cons' operator
10 favNums = 2:7:21:66:[]
11 morePrime2 = 2:morePrime
12
13 --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..]

```

A blue arrow points from the text "another example of laziness; although infinite list, check up to 20" in the red-bordered box to the line of code "evenUpto20 = takeWhile (<=20) [2,4..]" in the Notepad++ window.

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

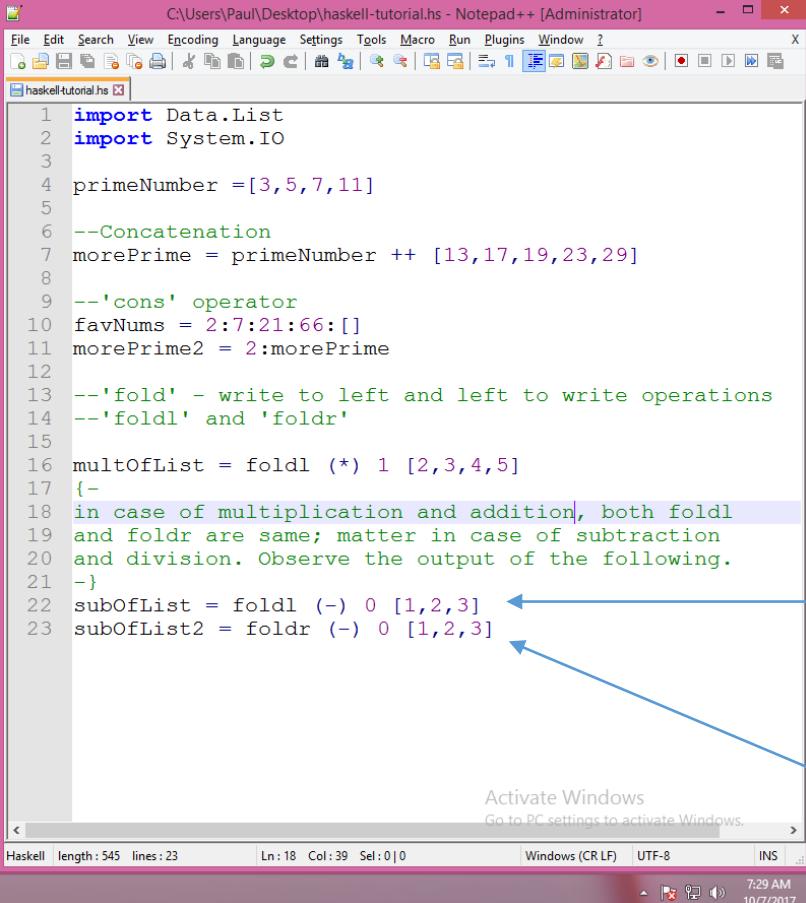
# More Operations on List



```

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> multOfList
120
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> subOfList
-6
*Main> subOfList2
2
*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 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] ←

Activate Windows
Go to PC settings to activate Windows.

Haskell | length: 545 lines: 23 | Ln: 18 Col: 39 Sel: 0 | 0 | Windows (CR LF) | UTF-8 | INS .:| 7:29 AM | 10/7/2017 |

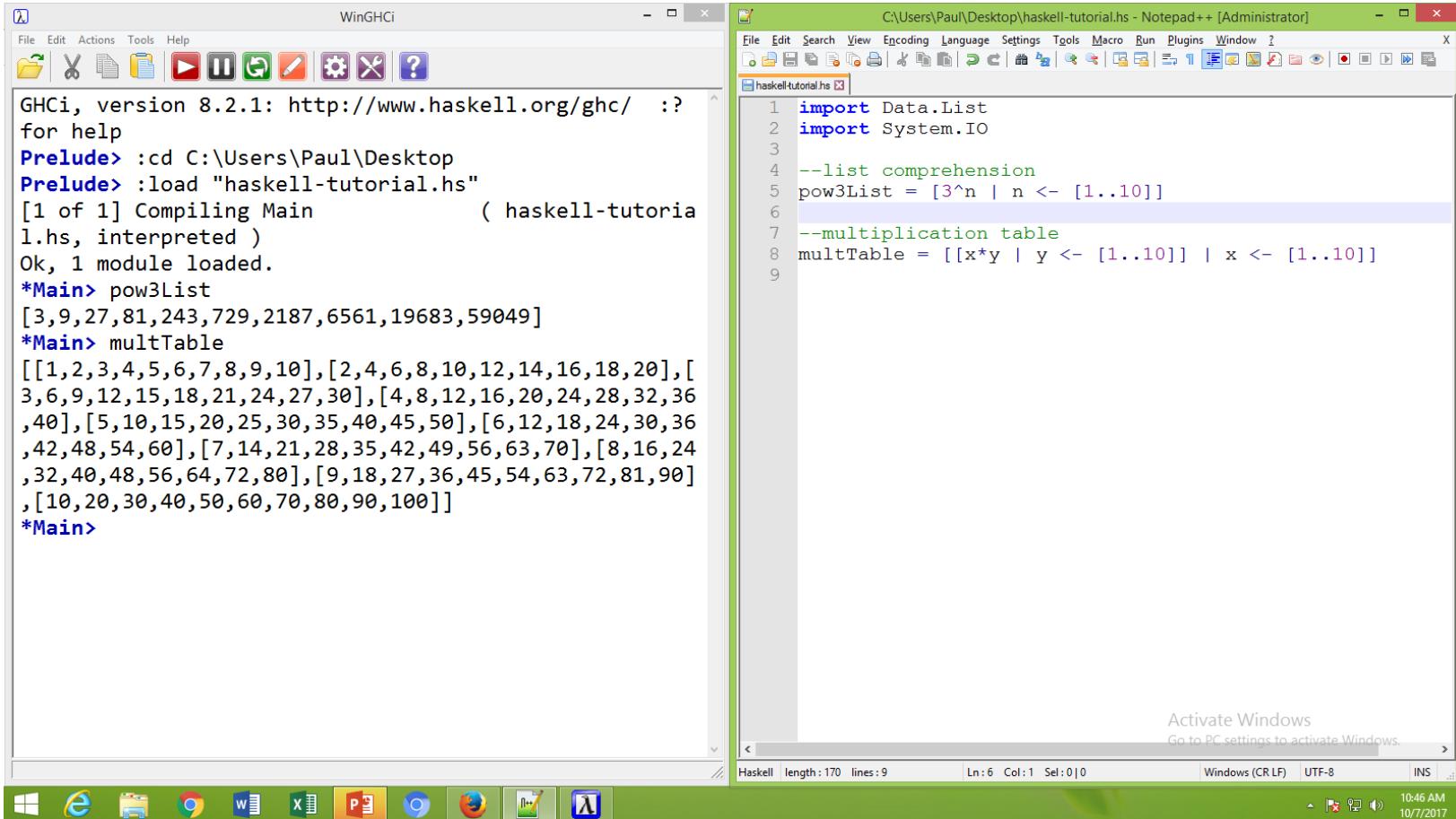
```

**foldl & foldr**

$((0-1)-2)-3)$

$1-(2-(3-0))$

# More Operations on List



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

- WinGHCi**: A terminal window for the Haskell interpreter (GHCi). 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> pow3List
[3,9,27,81,243,729,2187,6561,19683,59049]
*Main> multTable
[[1,2,3,4,5,6,7,8,9,10],[2,4,6,8,10,12,14,16,18,20],[3,6,9,12,15,18,21,24,27,30],[4,8,12,16,20,24,28,32,36,40],[5,10,15,20,25,30,35,40,45,50],[6,12,18,24,30,36,42,48,54,60],[7,14,21,28,35,42,49,56,63,70],[8,16,24,32,40,48,56,64,72,80],[9,18,27,36,45,54,63,72,81,90],[10,20,30,40,50,60,70,80,90,100]]
*Main>

```

- C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]**: A code editor window containing Haskell code:

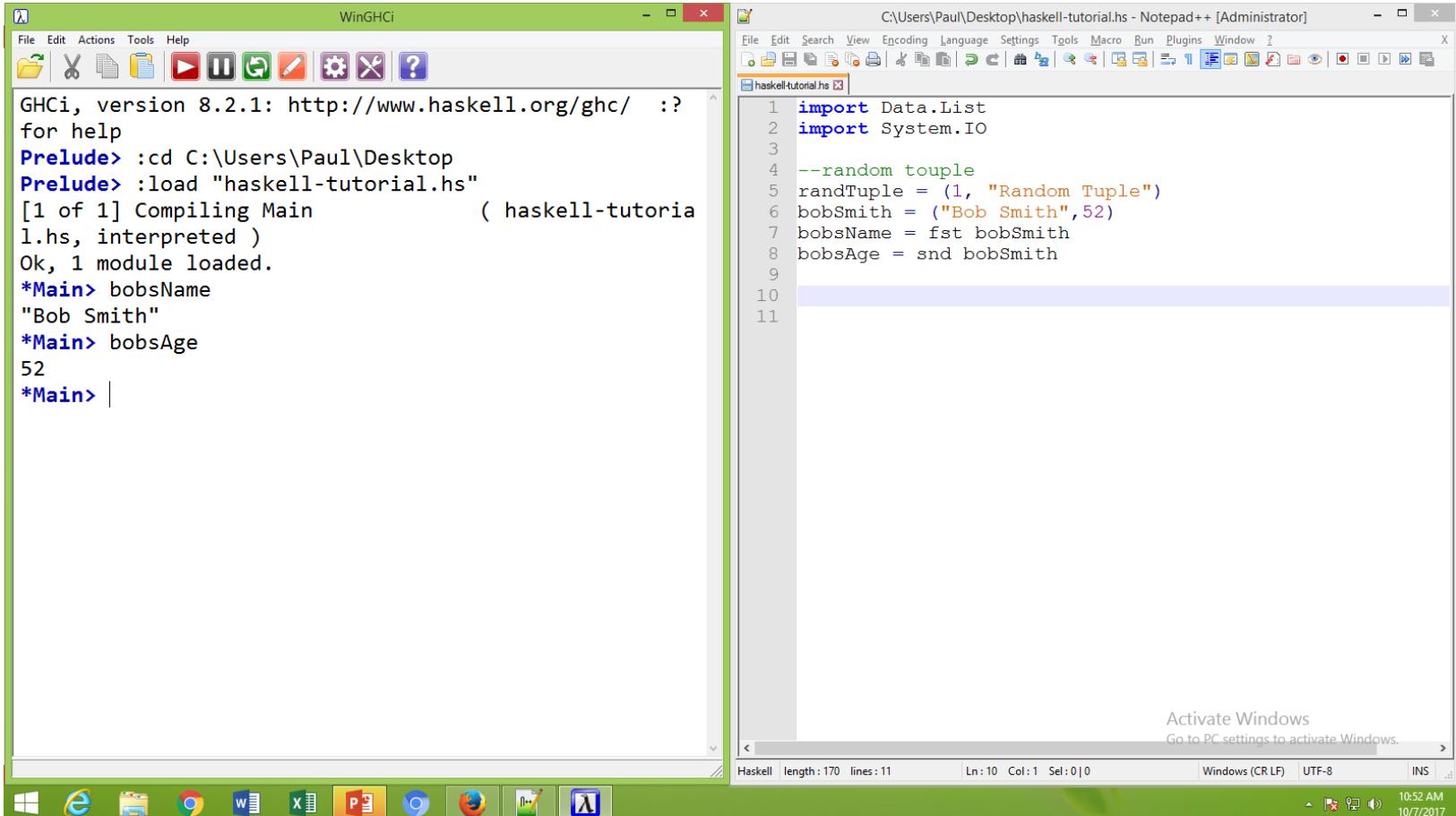
```

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

```

The taskbar at the bottom shows icons for various Windows applications like File Explorer, Edge, and Office.

# Multiple Data Type



The image shows two windows demonstrating Haskell code execution. On the left, the WinGHCi window displays the GHCi prompt and the execution of a Haskell script. On the right, the Notepad++ window shows the source code of the script.

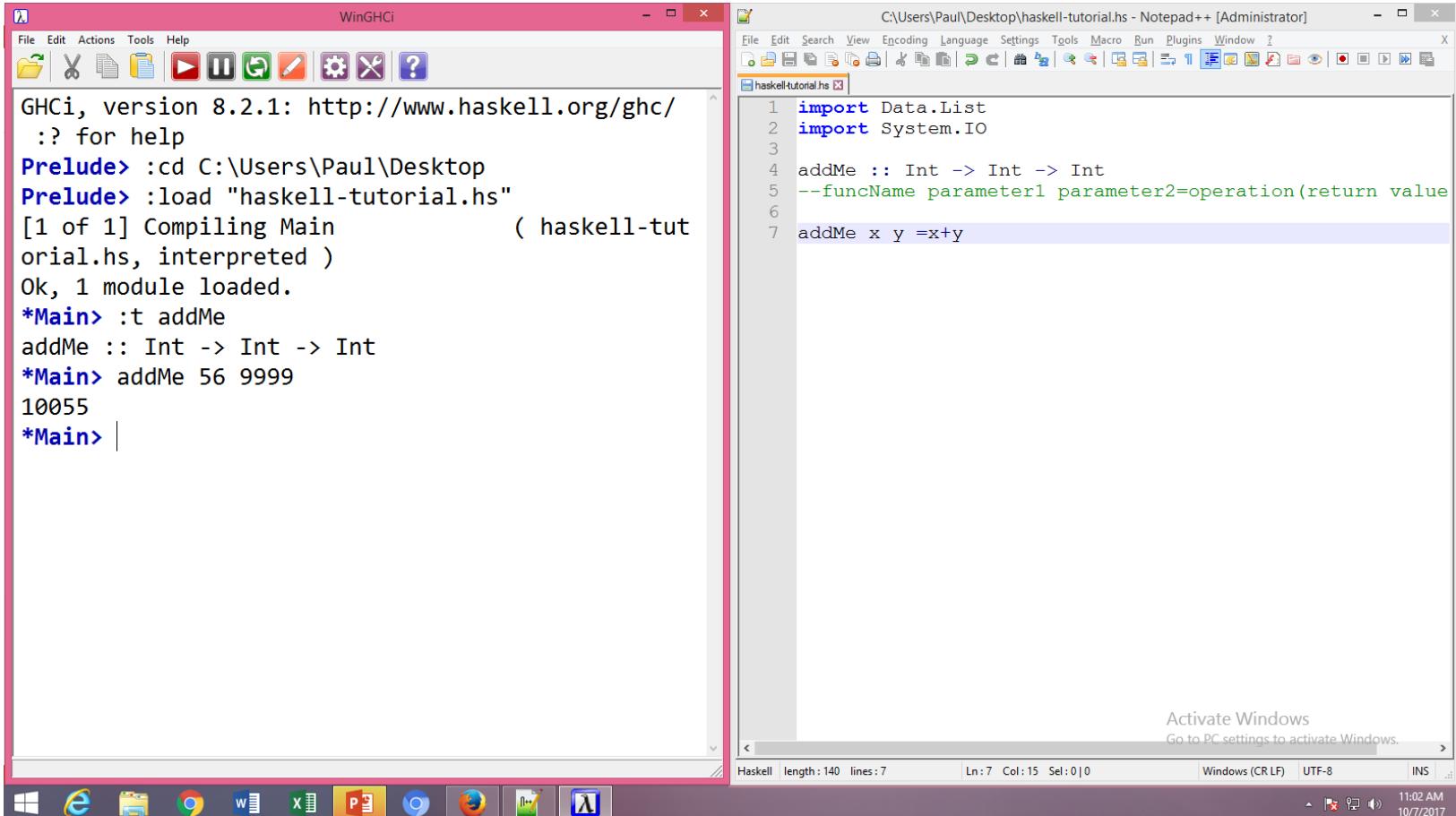
**WinGHCi Window:**

```
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> |
```

**Notepad++ Window:**

```
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



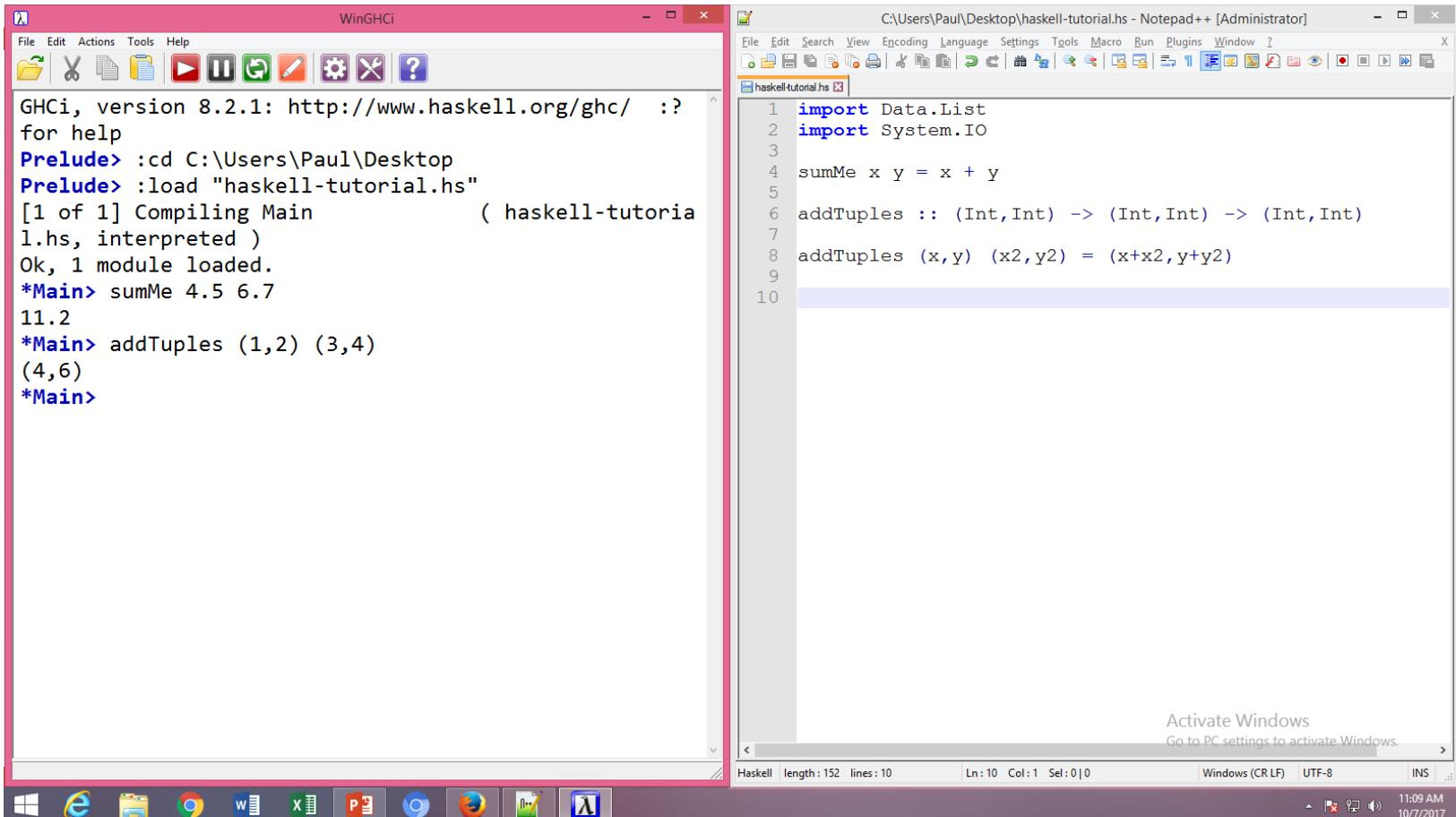
The image shows two windows demonstrating a Haskell function declaration. On the left, the WinGHCi window displays the interaction:

```
GHCi, version 8.2.1: http://www.haskell.org/ghc/
:?
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> :t addMe
addMe :: Int -> Int -> Int
*Main> addMe 56 9999
10055
*Main>
```

On the right, the Notepad++ window shows the source code of `haskell-tutorial.hs`:

```
1 import Data.List
2 import System.IO
3
4 addMe :: Int -> Int -> Int
5 --funcName parameter1 parameter2=operation(return value)
6
7 addMe x y =x+y
```

# User Type Declaration



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

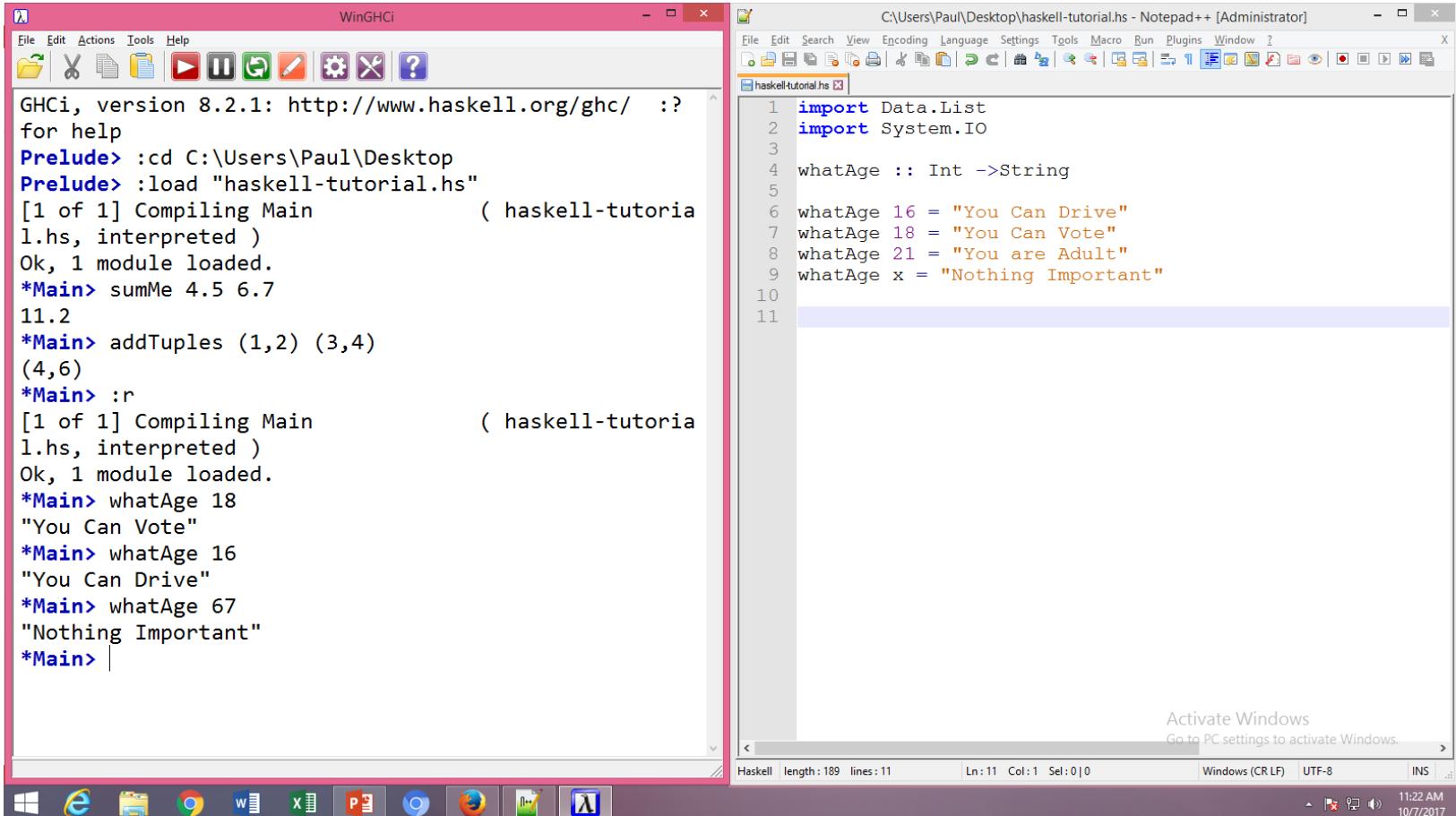
```
GHCi, version 8.2.1: http://www.haskell.org/ghc/  :?
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> sumMe 4.5 6.7
11.2
*Main> addTuples (1,2) (3,4)
(4,6)
*Main>
```

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

```
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
```

The status bar at the bottom of the Notepad++ window indicates the file is in Haskell mode, with a length of 152 and 10 lines. The system tray shows the date and time as 10/7/2017 and 11:09 AM.

# User Type Declaration



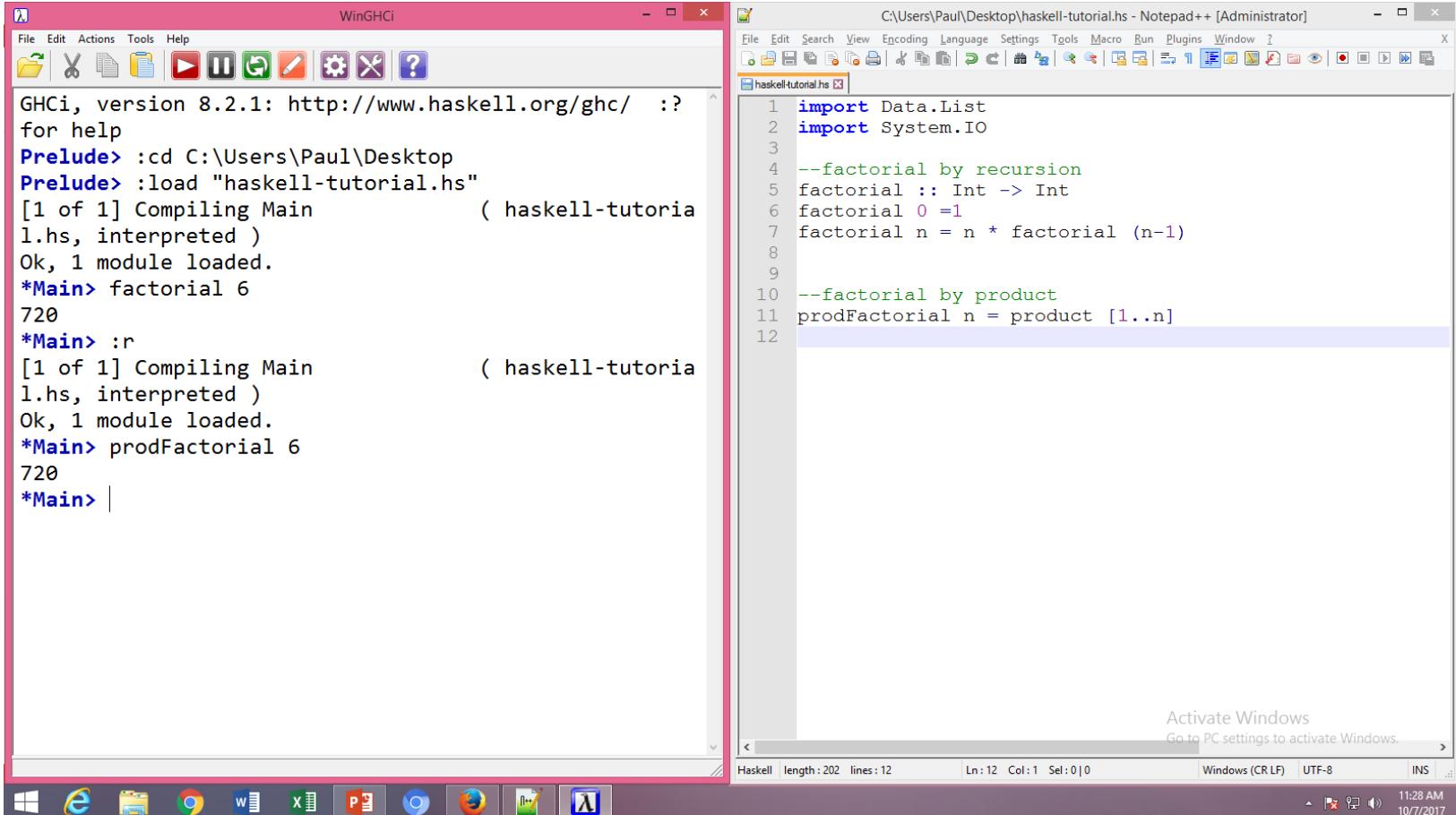
```
WinGHCi
File Edit Actions Tools Help
GHCi, version 8.2.1: http://www.haskell.org/ghc/  ?: 
for help
Prelude> :cd C:\Users\Paul\Desktop
Prelude> :load "haskell-tutorial.hs"
[1 of 1] Compiling Main           ( haskell-tutorial
1.hs, interpreted )
Ok, 1 module loaded.
*Main> sumMe 4.5 6.7
11.2
*Main> 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]

```
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
```

Haskell | length: 189 | lines: 11 | Ln: 11 Col: 1 Sel: 0 | 0 | Windows (CR LF) | UTF-8 | INS | .: | 11:22 AM | 10/7/2017 |

# Factorial (by recursion and by product)



**WinGHCi**

```

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>

```

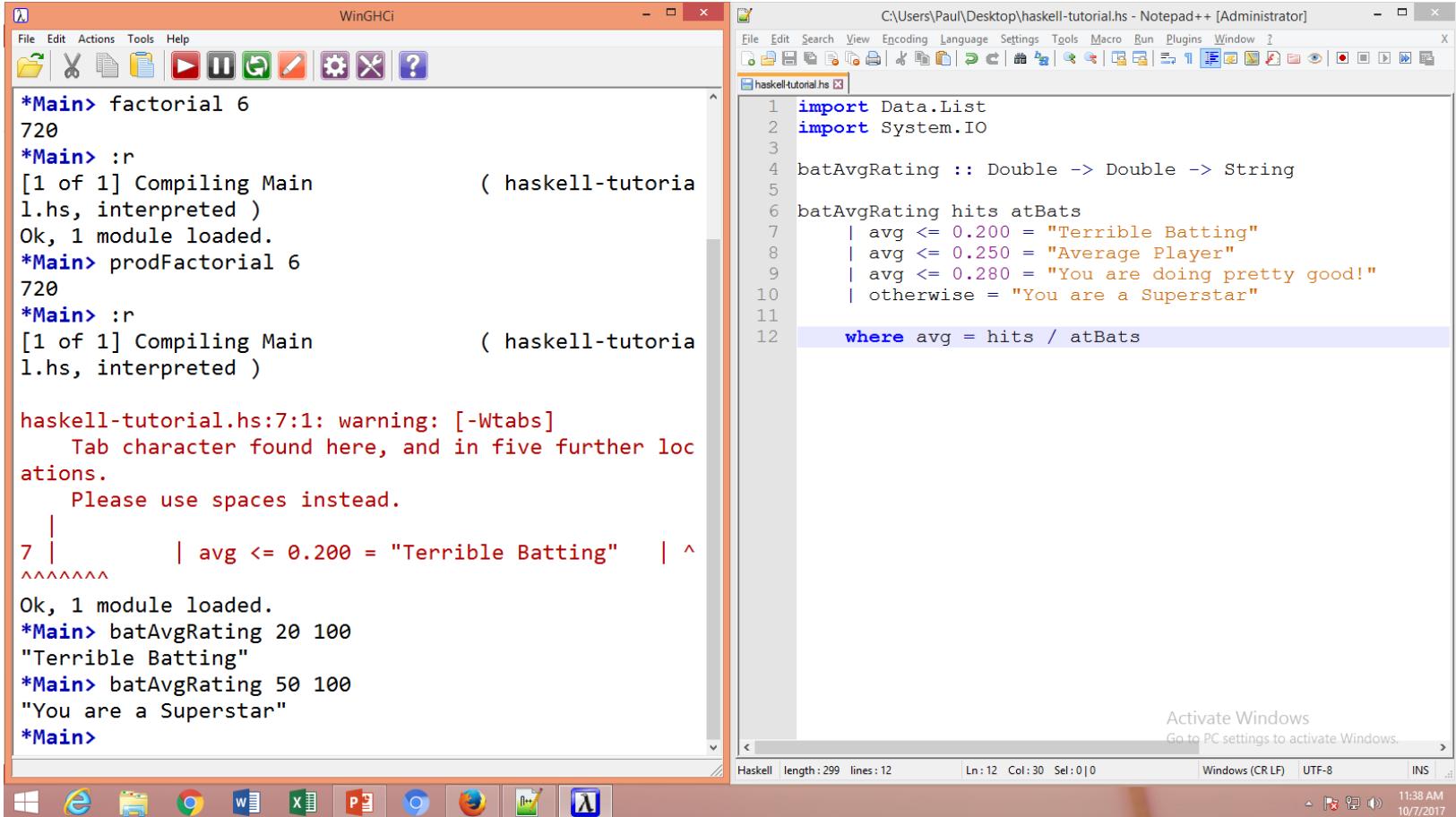
**C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]**

```

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:

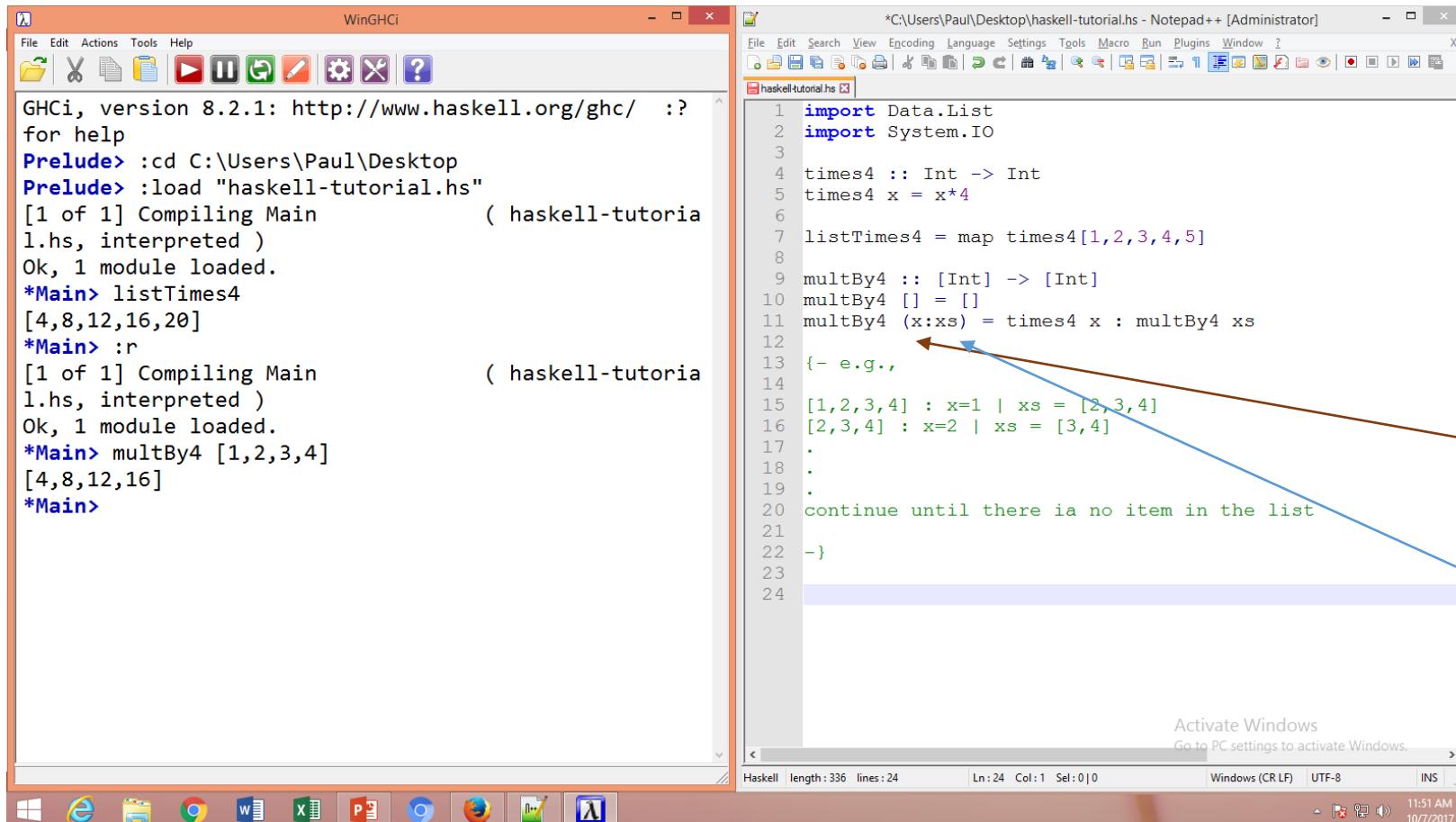
```
*Main> factorial 6
720
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutoria
1.hs, interpreted )
Ok, 1 module loaded.
*Main> prodFactorial 6
720
*Main> :r
[1 of 1] Compiling Main           ( haskell-tutoria
1.hs, interpreted )

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 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> 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, which contains the Haskell code for the `haskell-tutorial.hs` file. The code defines two functions: `times4` and `multBy4`. The `multBy4` function uses `map` to apply `times4` to each element of a list. A red box highlights the part of the code where `multBy4` is defined, and a blue arrow points from this code to a text box on the right.

```

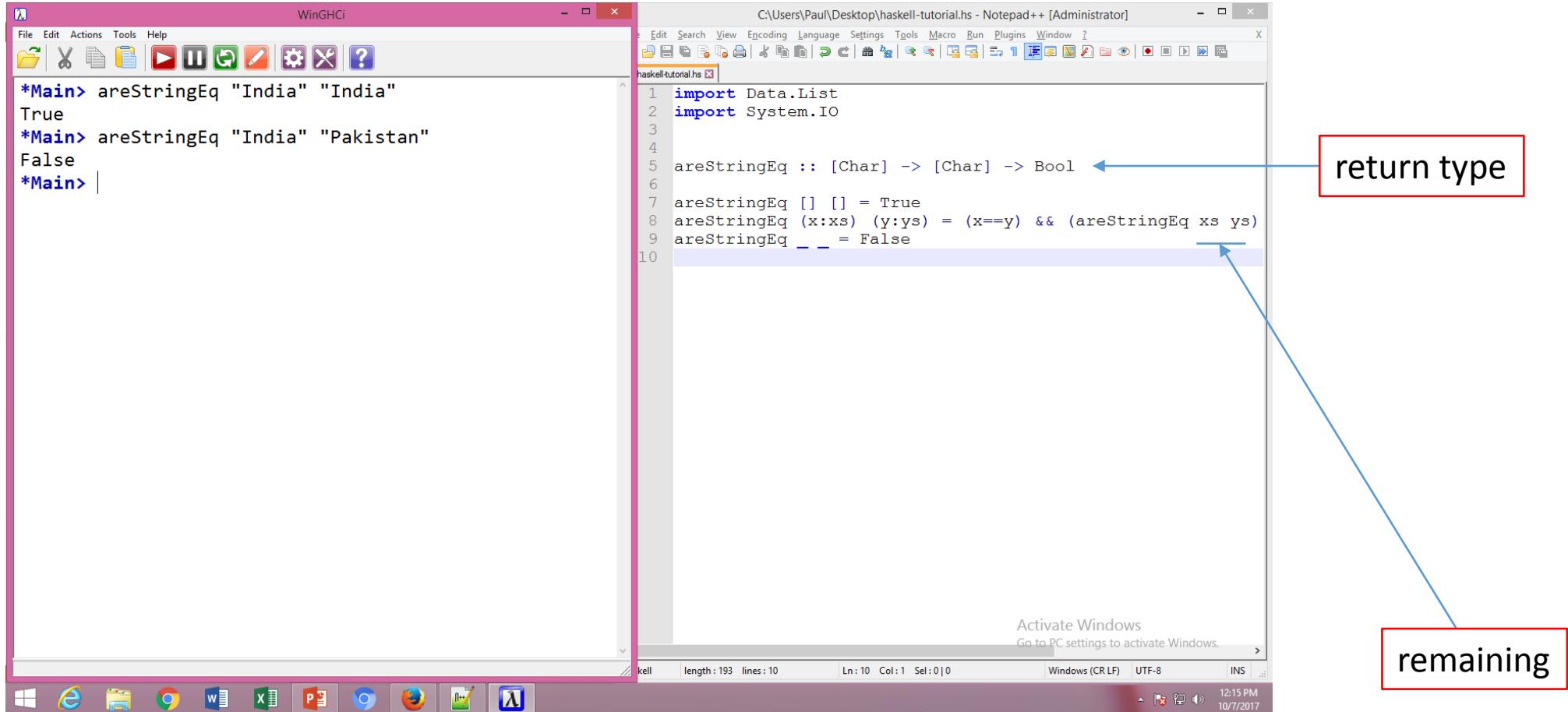
*C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
haskell-tutorial.hs x
1 import Data.List
2 import System.IO
3
4 times4 :: Int -> Int
5 times4 x = x*4
6
7 listTimes4 = map times4[1,2,3,4,5]
8
9 multBy4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

Activate Windows
Go to PC settings to activate Windows.
Haskell | length: 336 lines: 24 | Ln: 24 Col: 1 Sel: 0 | 0 | Windows (CR LF) | UTF-8 | INS | . .
1:51 AM 10/7/2017

```

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 REPL session:

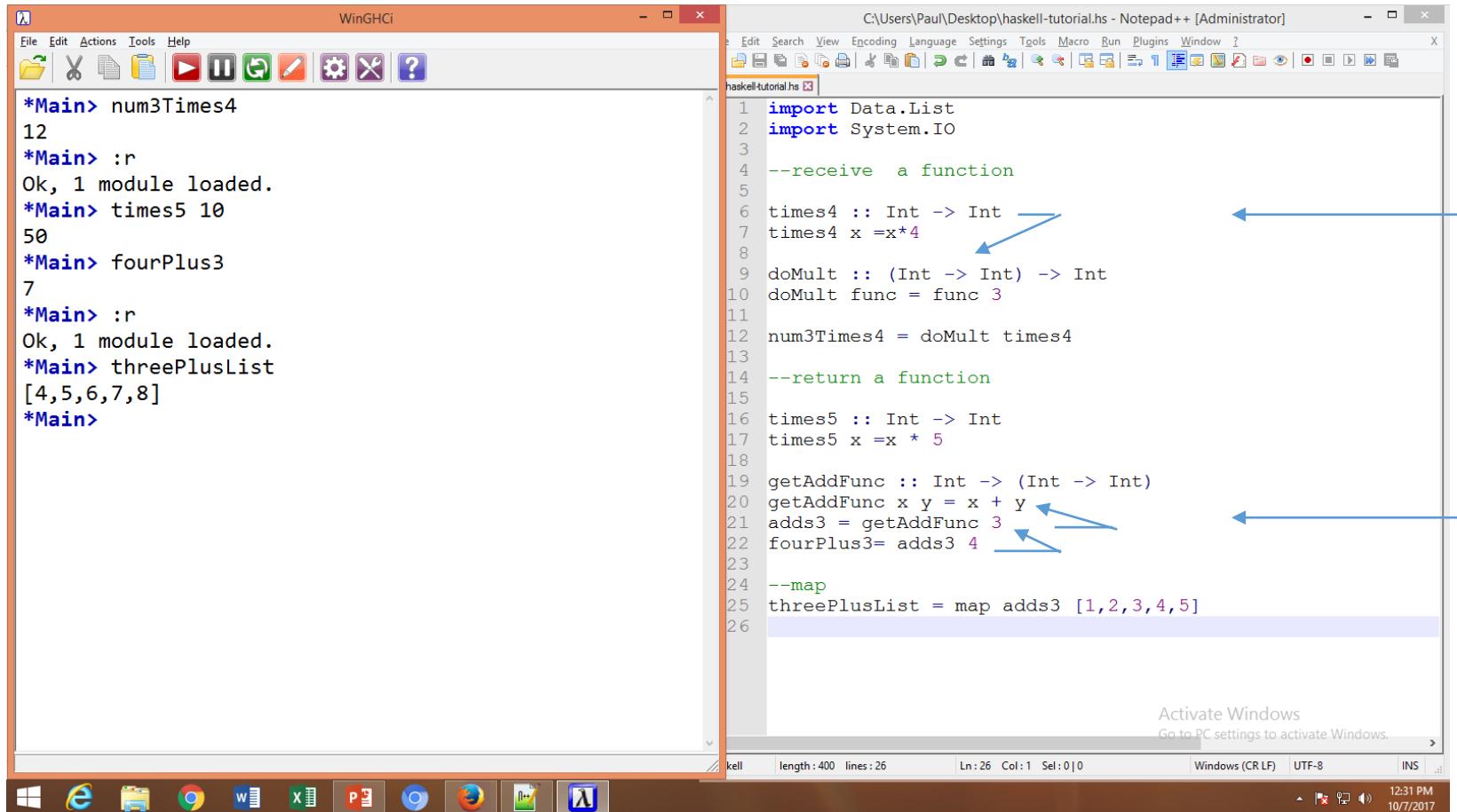
```
*Main> areStringEq "India" "India"
True
*Main> areStringEq "India" "Pakistan"
False
*Main>
```

On the right is the Notepad++ window, showing the source code for the `areStringEq` function:

```
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
9
10
```

A red box labeled "return type" highlights the type signature `[Char] -> [Char] -> Bool`. A blue arrow points from this label to the type signature. Another blue arrow points from the label "remaining" to the final line of the function definition.

# Receive and Return a Function



The image shows two windows side-by-side. On the left is the WinGHCi window, which contains a Haskell REPL session. On the right is a Notepad++ window containing Haskell code.

**WinGHCi Window:**

```
*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>
```

**Notepad++ Window (haskell-tutorial.hs):**

```
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 adds3 = getAddFunc 3
22 fourPlus3= adds3 4
23
24 --map
25 threePlusList = map adds3 [1,2,3,4,5]
```

Annotations in the Notepad++ window highlight specific parts of the code:

- A red box labeled "receive" with a blue arrow points to the line `times4 :: Int -> Int`.
- A red box labeled "return" with a blue arrow points to the line `getAddFunc :: Int -> (Int -> Int)`.

# Other Operators

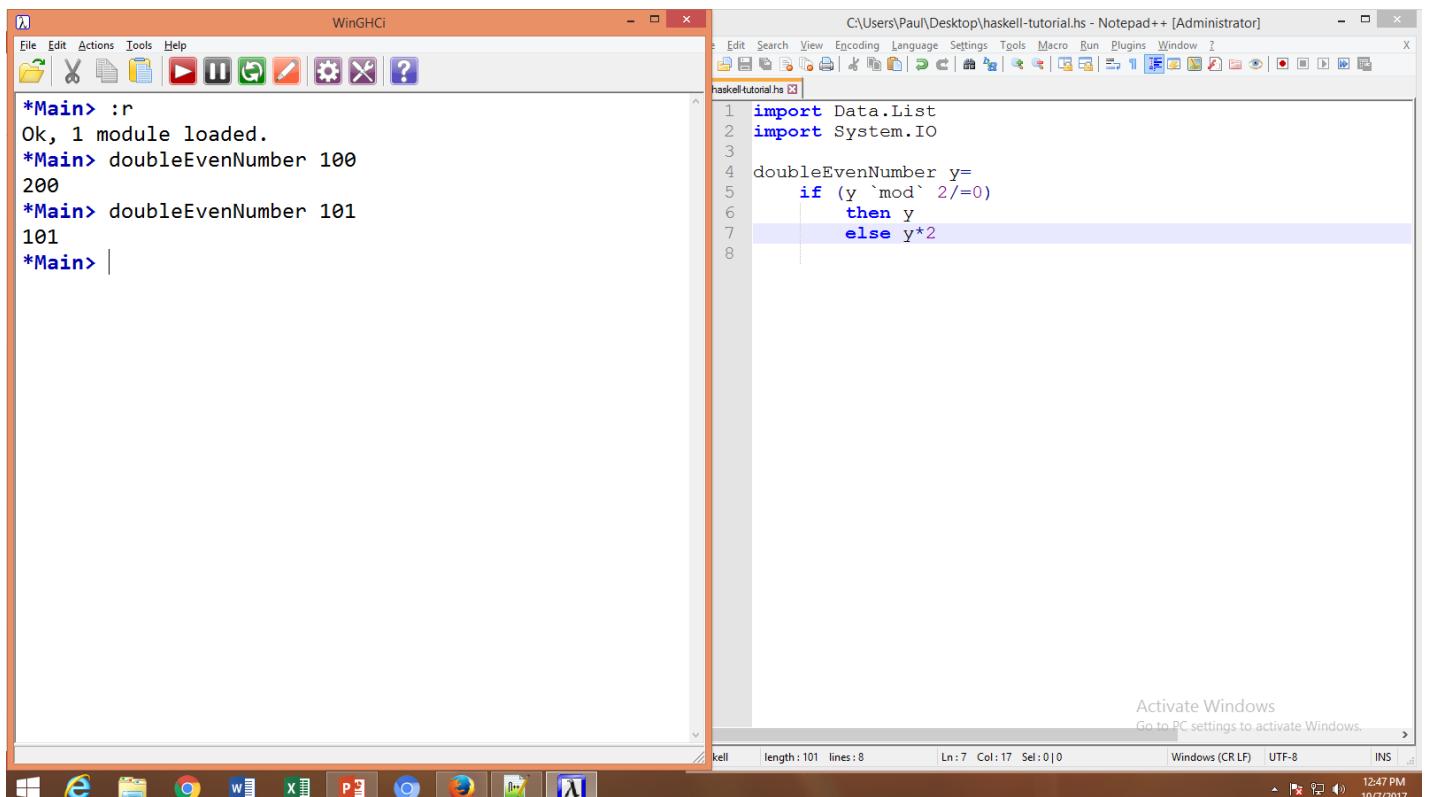
## ➤ Comparison

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

## ➤ Logical

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

## Example

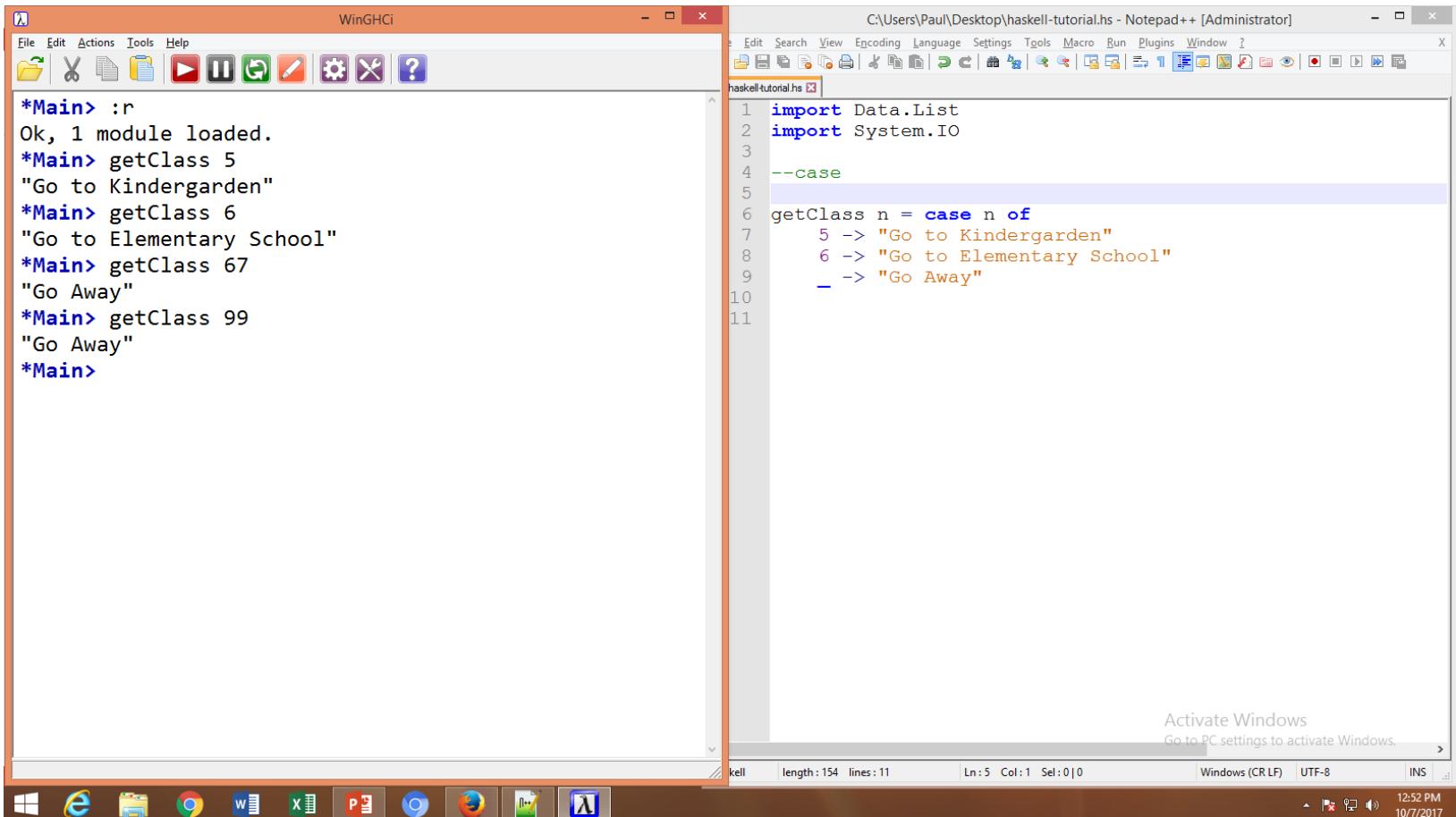


The screenshot shows a Windows desktop environment. In the center, there is a terminal window titled "WinGHCi" which displays Haskell code and its execution results. To the right of the terminal is a code editor window titled "haskell-tutorial.hs" containing Haskell code. The Haskell code defines a function "doubleEvenNumber" that takes an integer y and returns y if it is odd or y\*2 if it is even. The terminal shows the loading of the module and the execution of the function with arguments 100 and 101.

```
*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
```

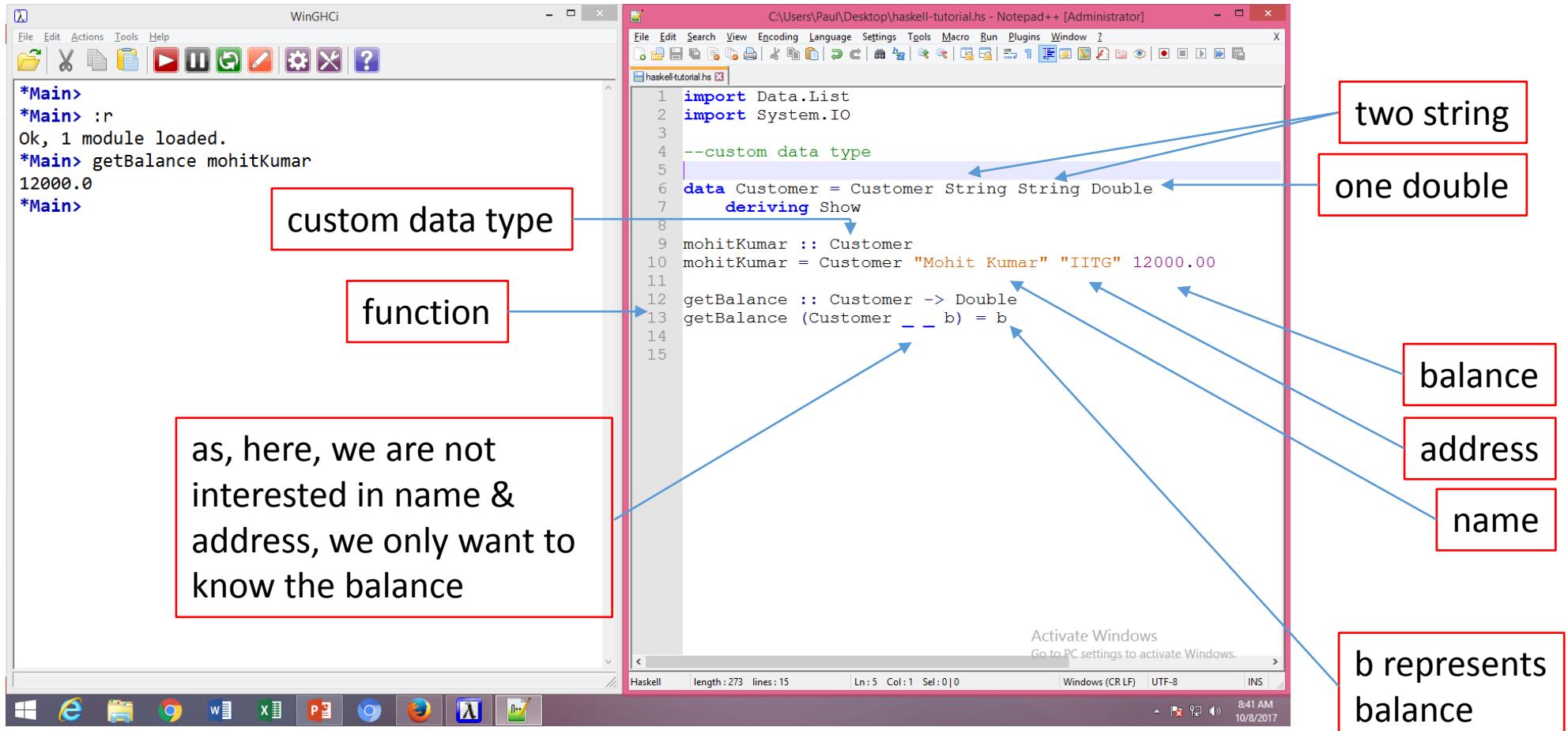
# Case



The image shows two windows side-by-side. On the left is the WinGHCi window, which contains a Haskell REPL session. The session starts with ':r' to load a module, followed by several calls to 'getClass' with different arguments (5, 6, 67, 99) and a 'getAway' call. The responses are "Go to Kindergarten", "Go to Elementary School", "Go Away", and "Go Away" respectively. On the right is a Notepad++ window titled 'haskell-tutorial.hs' containing the following Haskell code:

```
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"
```

# Custom Data Type



The image shows two windows: WinGHCi on the left and Notepad++ on the right.

**WinGHCi Window:**

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

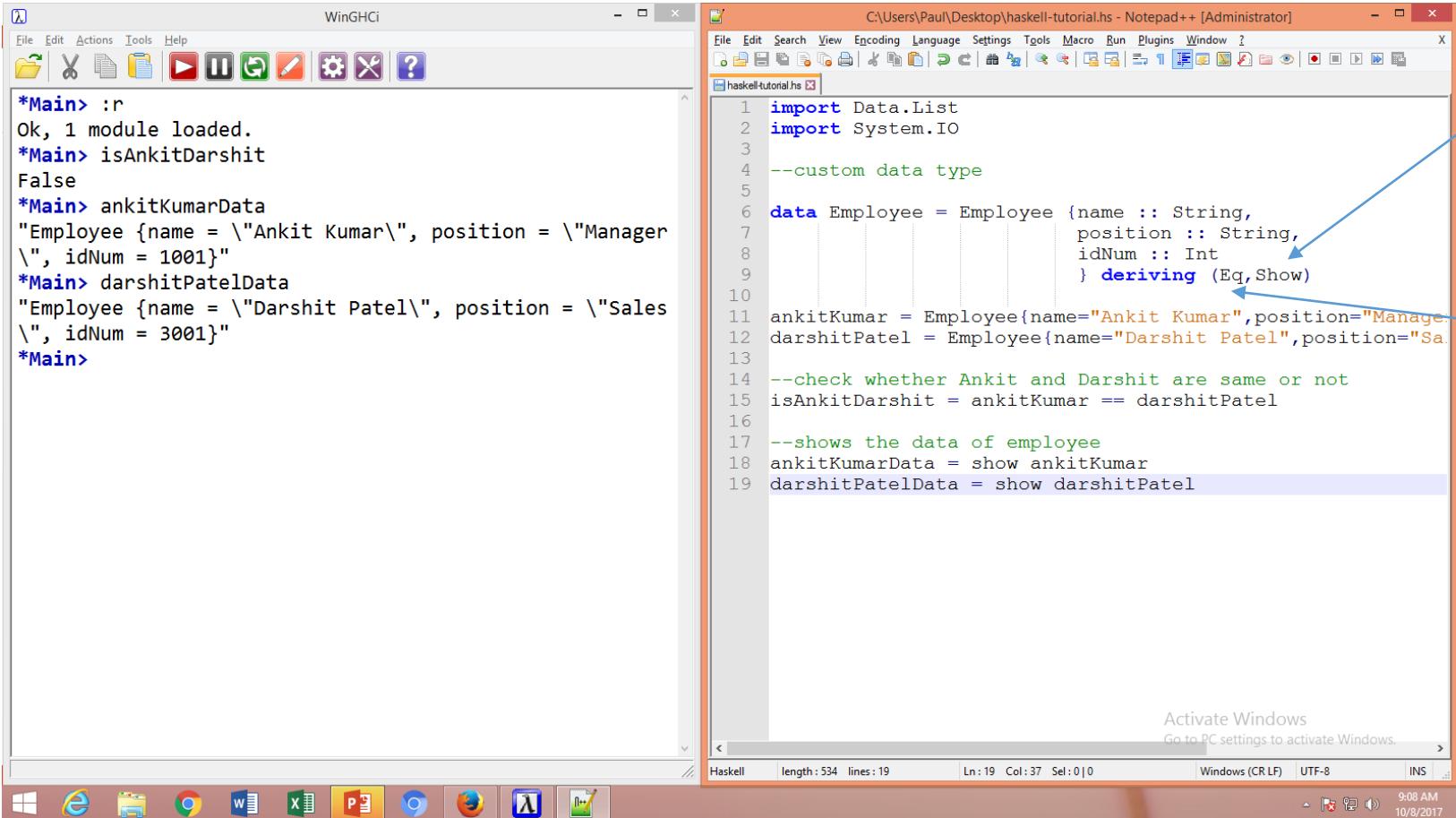
**Notepad++ Window (haskell-tutorial.hs):**

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

**Annotations:**

- custom data type**: Points to the line `data Customer = Customer String String Double`.
- function**: Points to the line `getBalance :: Customer -> Double`.
- as, here, we are not interested in name & address, we only want to know the balance**: Points to the line `getBalance (Customer _ _ b) = b`.
- two string**: Points to the first two fields of the `Customer` constructor.
- one double**: Points to the third field of the `Customer` constructor.
- balance**: Points to the variable `b` in the `getBalance` function.
- address**: Points to the second field of the `Customer` constructor.
- name**: Points to the first field of the `Customer` constructor.
- b represents balance**: Points to the variable `b` in the `getBalance` function.

# Type Classes



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

```
*Main> :r
Ok, 1 module loaded.
*Main> isAnkitDarshit
False
*Main> ankitKumarData
"Employee {name = \"Ankit Kumar\", position = \"Manager\",
\", idNum = 1001}"
*Main> darshitPatelData
"Employee {name = \"Darshit Patel\", position = \"Sales\",
\", idNum = 3001}"
*Main>
```

On the right is a Notepad++ window containing Haskell code named "haskell-tutorial.hs". The code defines a custom data type "Employee" and performs some operations on it:

```
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
```

Two red boxes with arrows point from the text to the right of the code. The top box points to the "isAnkitDarshit" line and contains the text "able to check for the equality". The bottom box points to the "show" lines and contains the text "able to show the employee details".

---

# END OF TUTORIAL

YOU MAY EXPLORE

<http://www.learnyouahaskell.com>

FOR MORE DETAIL

---