Functional Programming Concept with

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

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

Lets Start

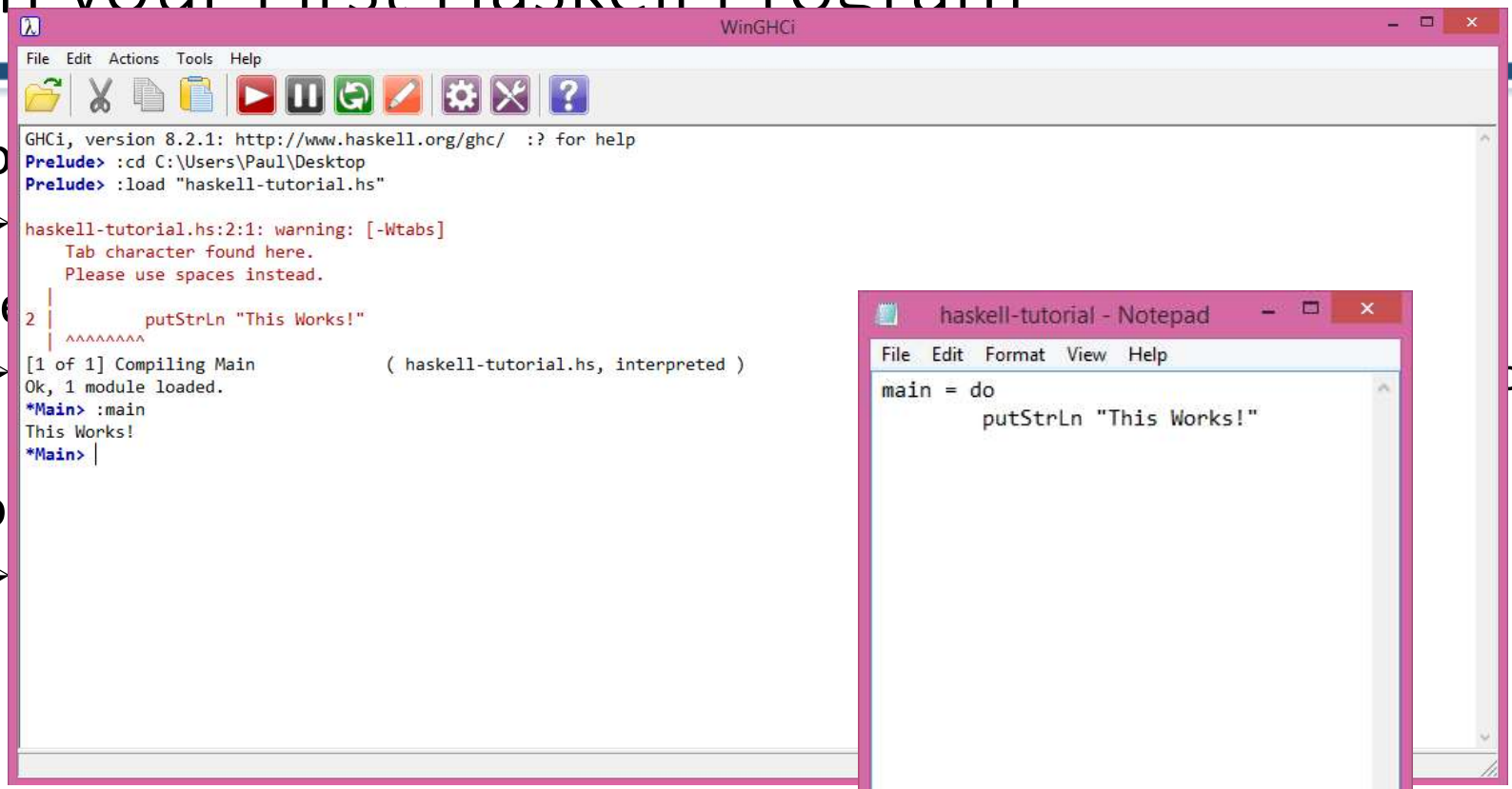
Lets try to understand basic features of Haskell with examples

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-tutorail.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

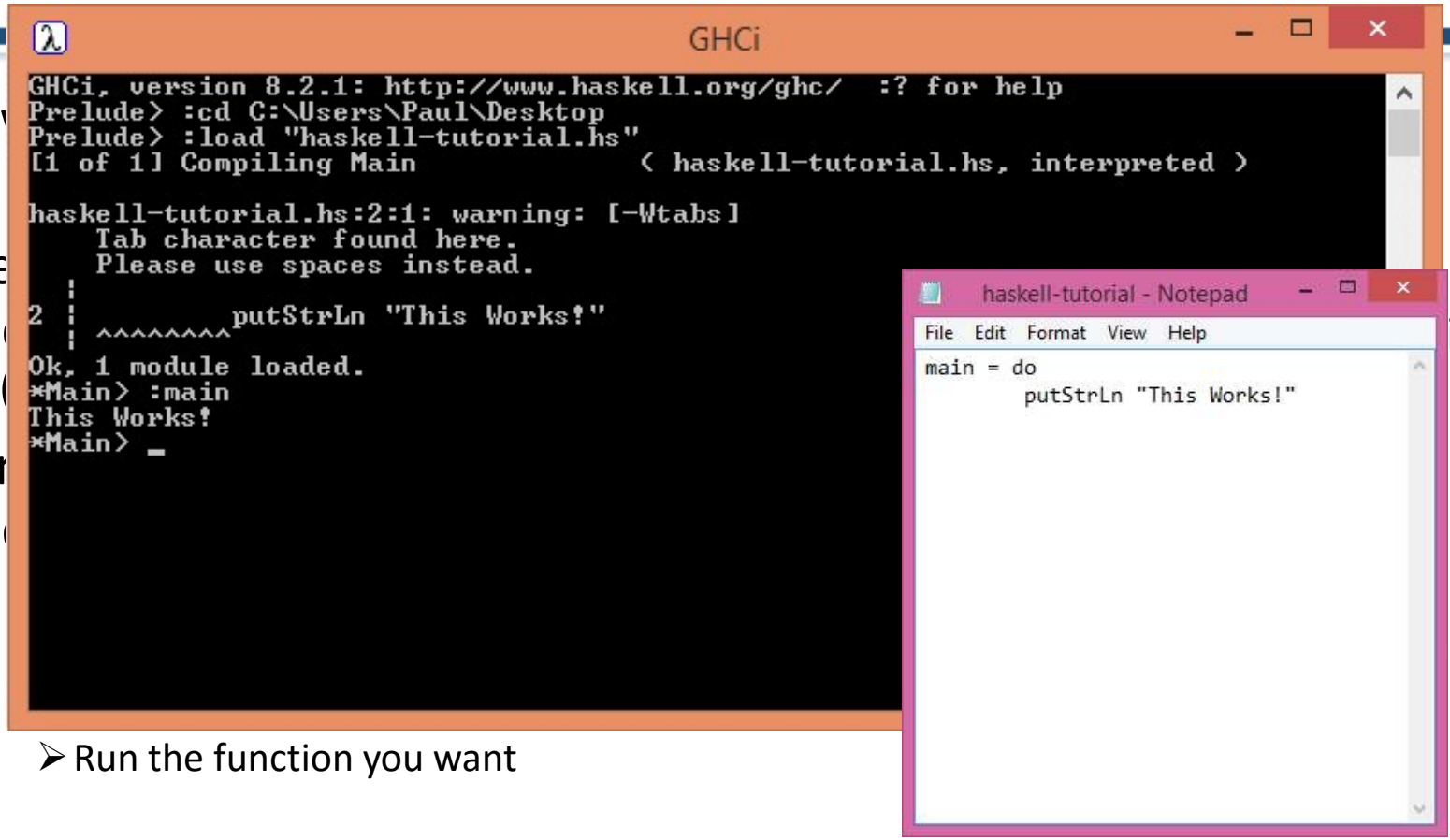
- Do
  -
- File
  -
- Co
  -



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-tutorail.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



- Do
  -
- File
  -
- Co
  -

ion

p)

- Run the function you want

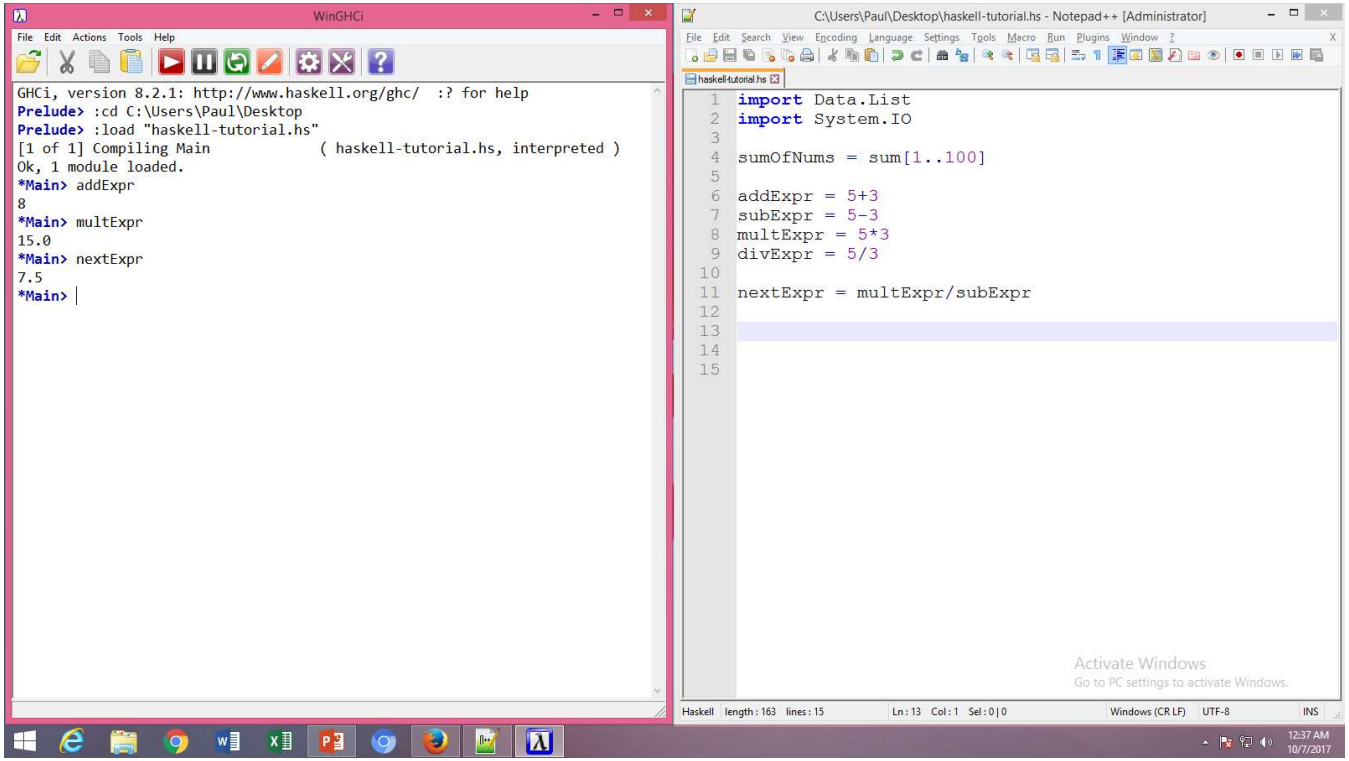
Few things you may keep in mind

- 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

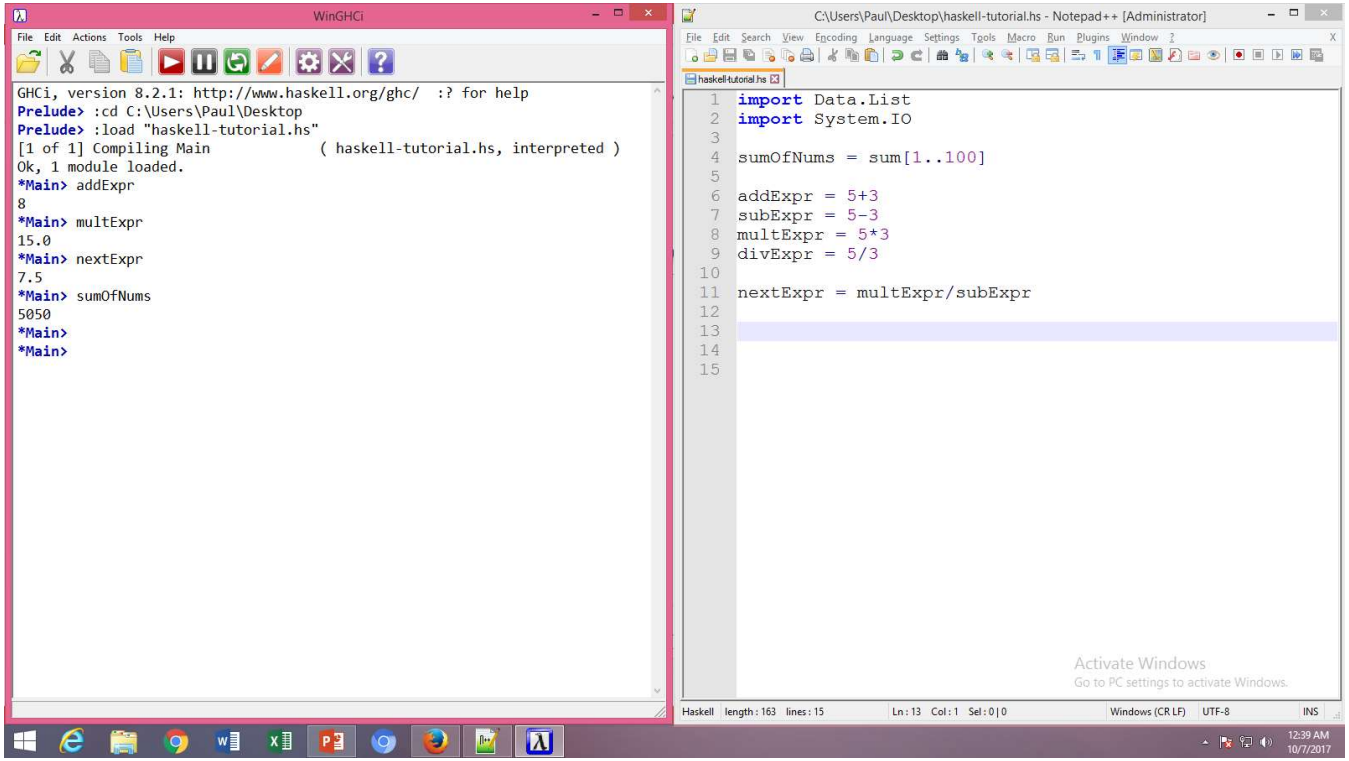
Date Types

- 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

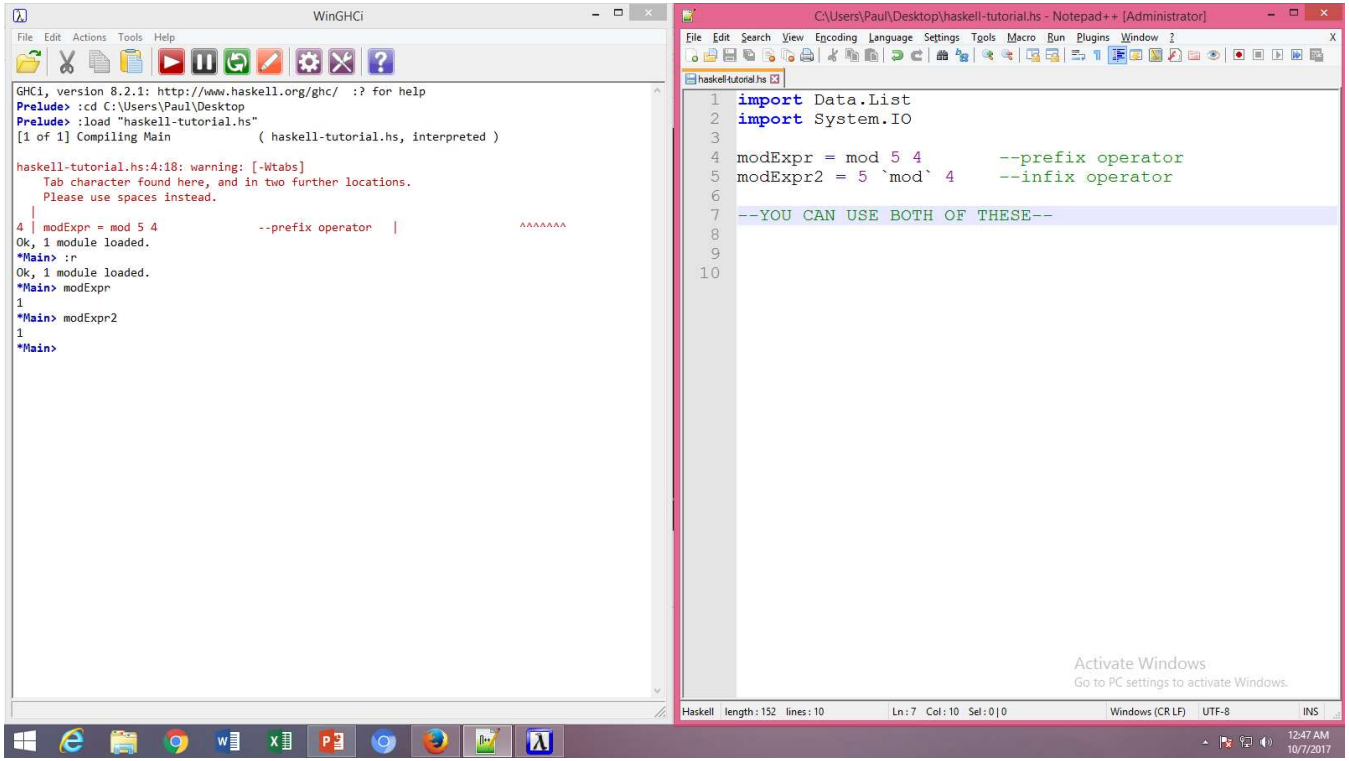
Expressions



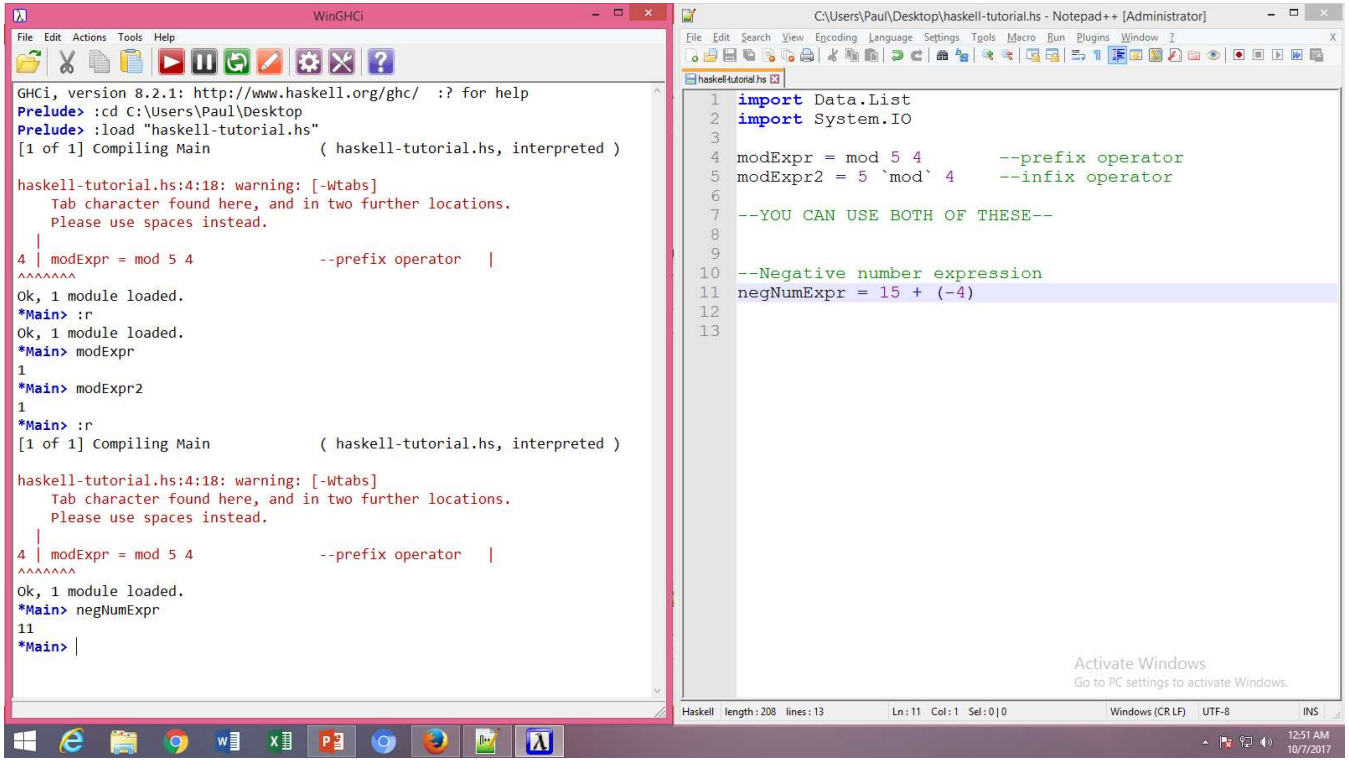
Expressions



Infix and Prefix Operator



Negative Number Expression

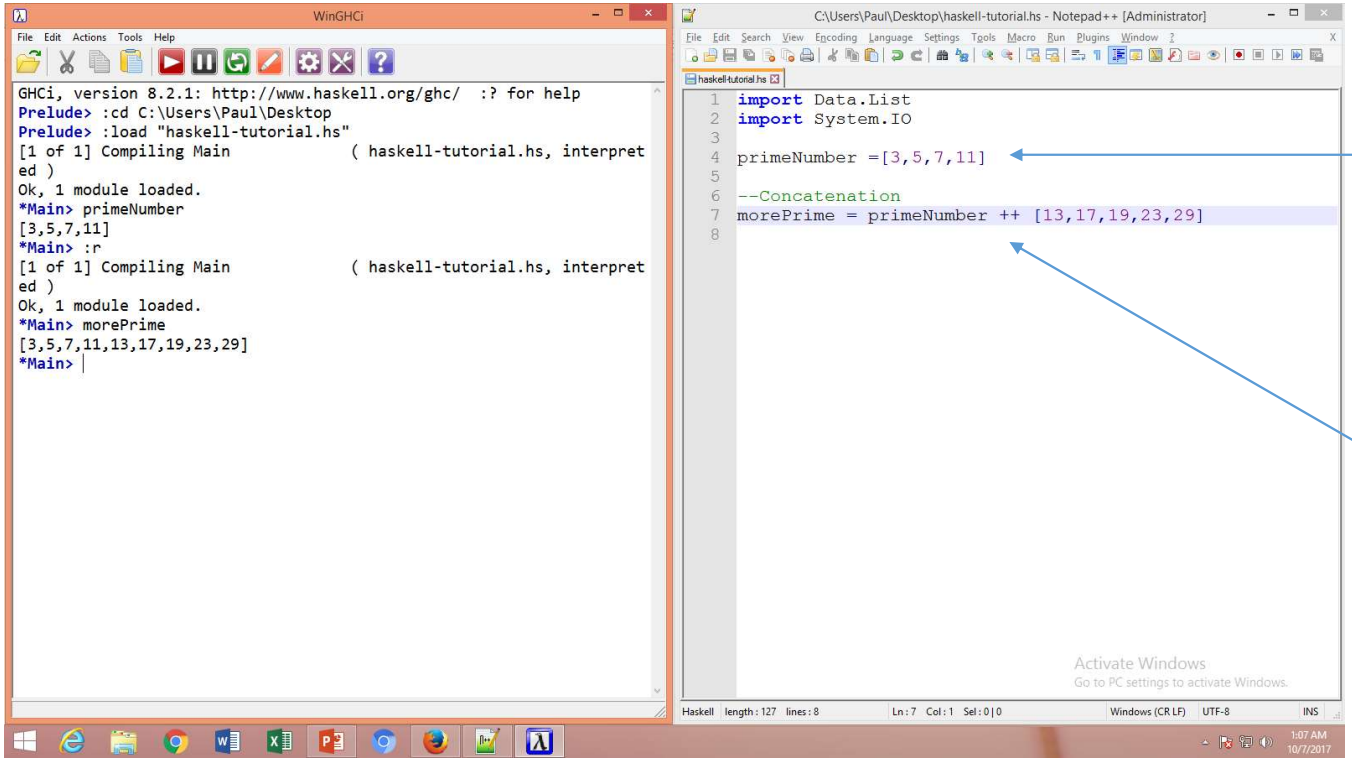


Other built-in Math Function

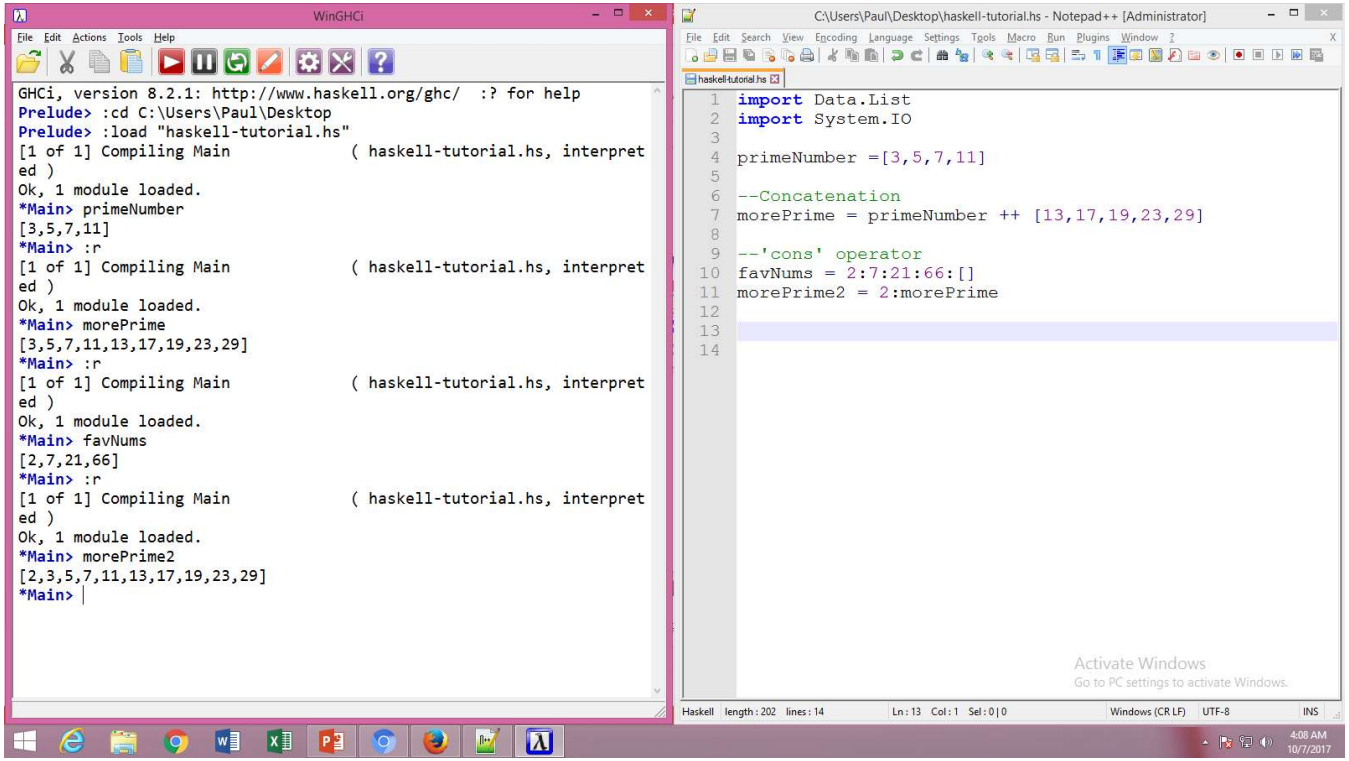
- piVal = pi
- ePow9 = exp 9
- logOf9 = log 9
- Squared9 = 9 ** 2
- truncateVal = truncate 9.999
- roundVal = round 9.999
- ceilingVal = ceiling 9.999
- floorVal = floor 9.999
- Also
  - sin, cos, tan, asign, acos, atan, signh, cosh, tanh, asignh, acosh, atanh

EXPLORE THESE

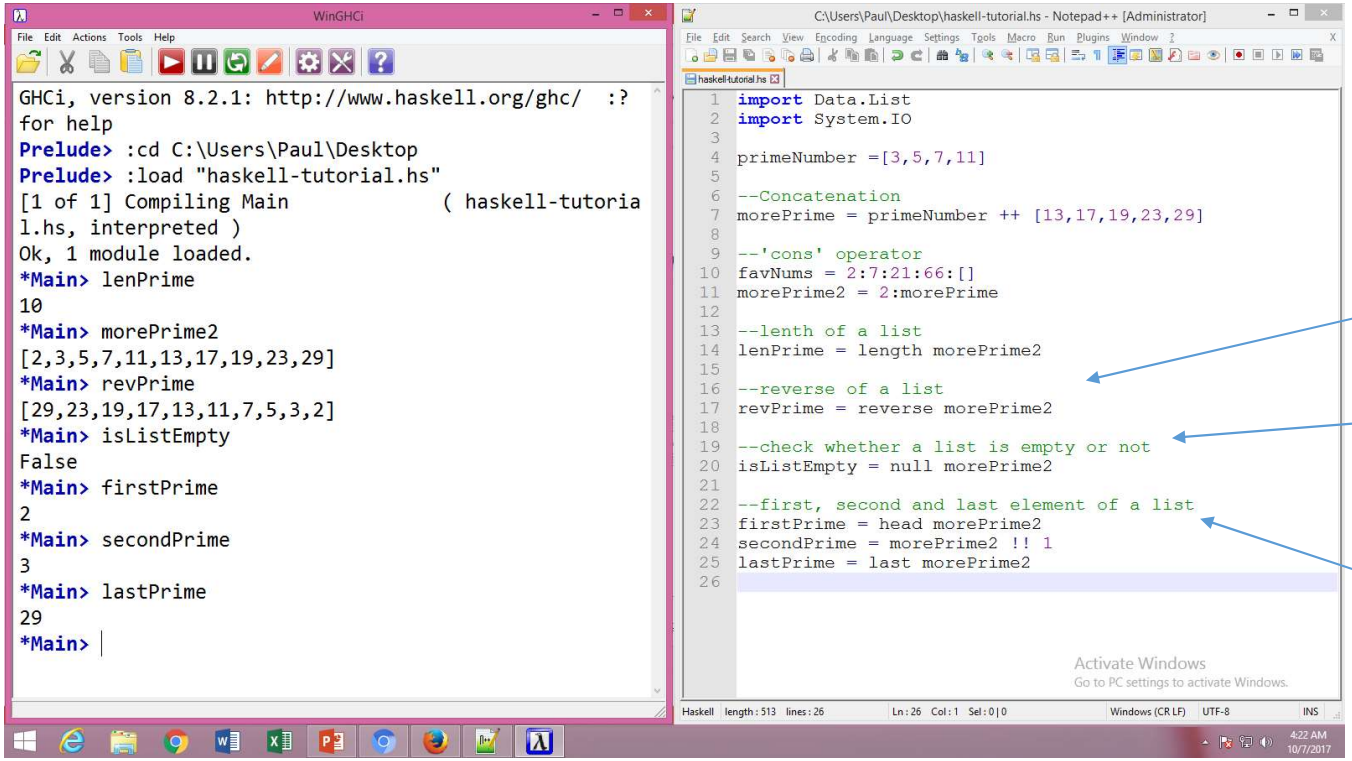
List - Concatenation



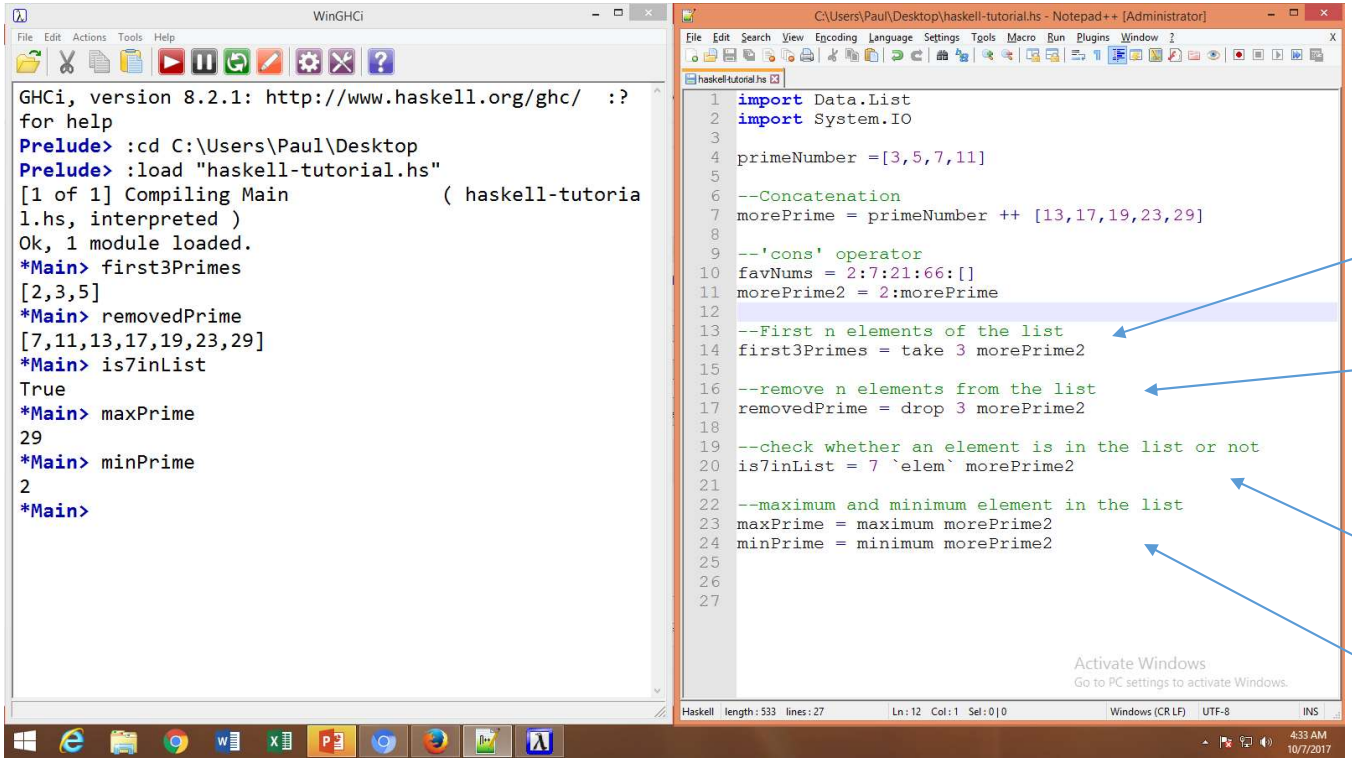
List – ‘cons’ operator



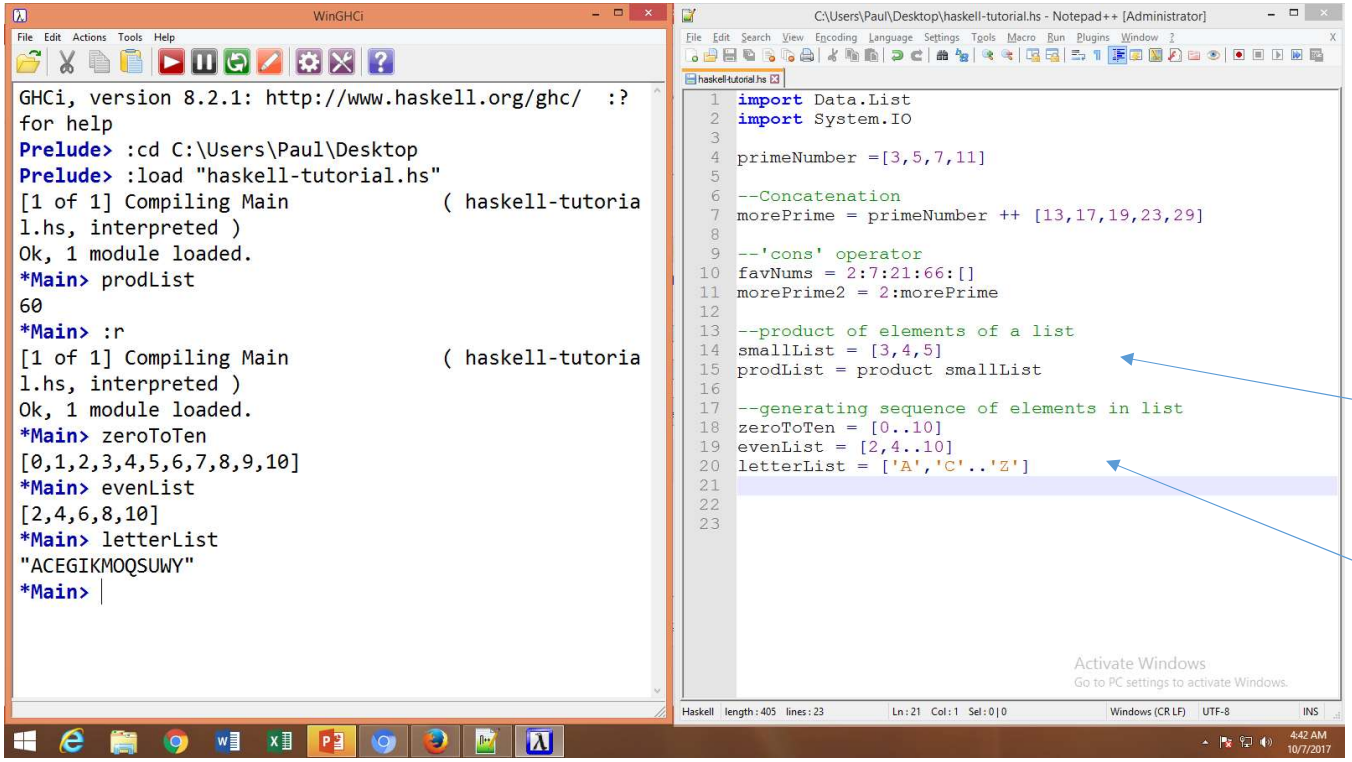
More Operations on List



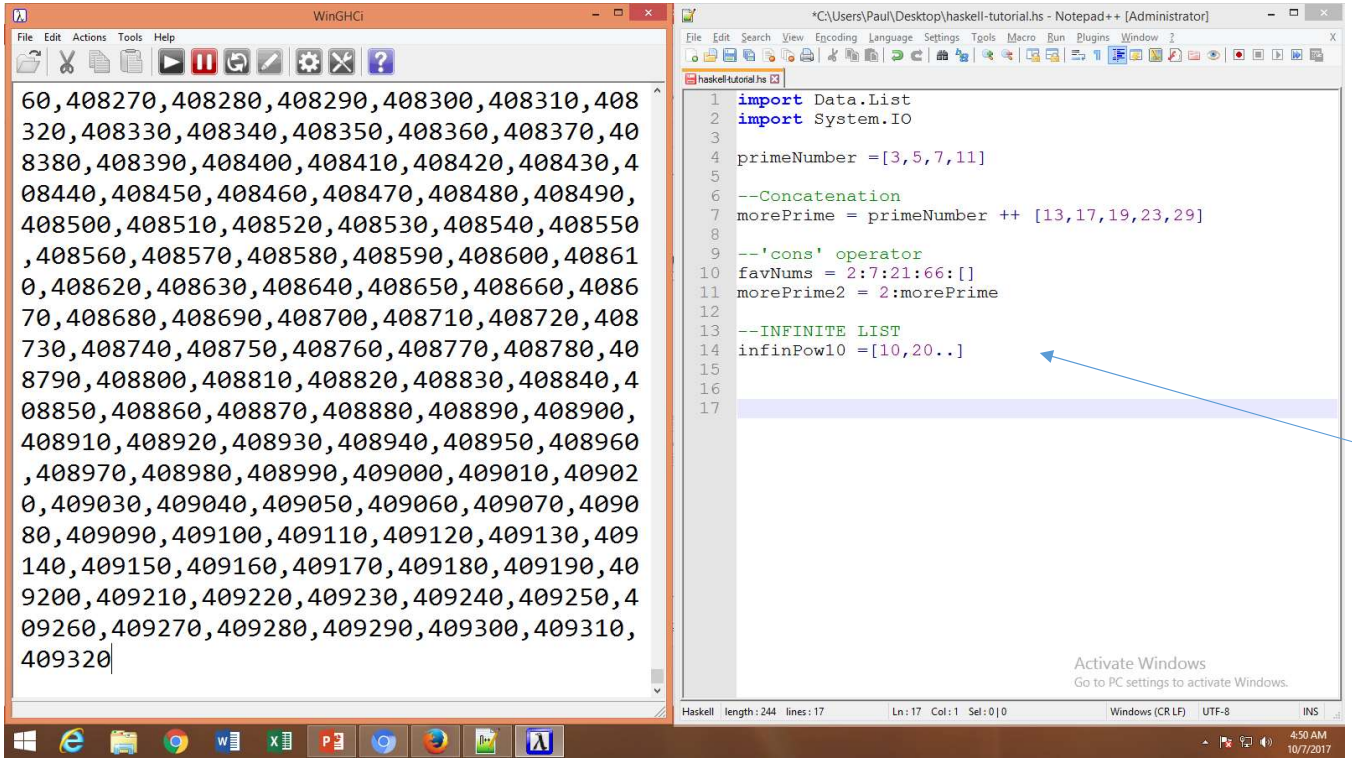
More Operations on List



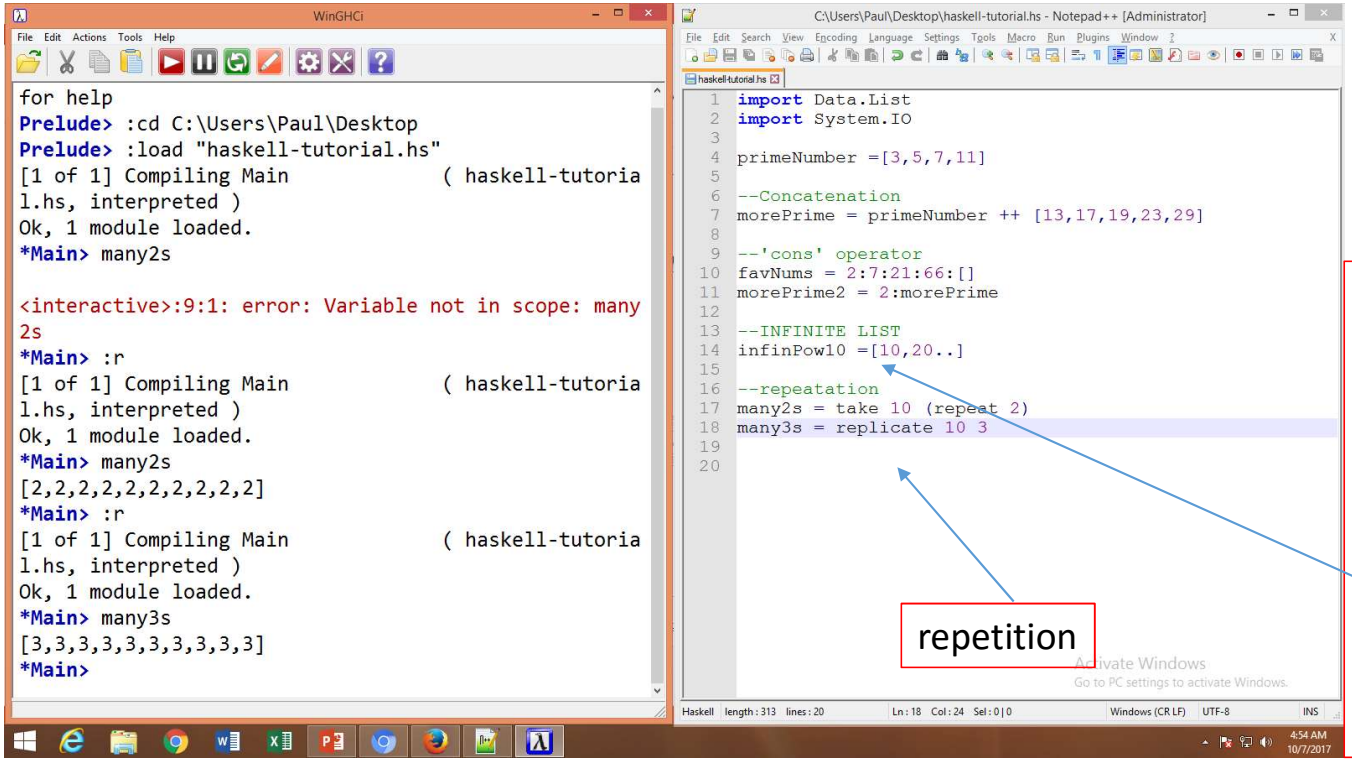
More Operations on List



More Operations on List

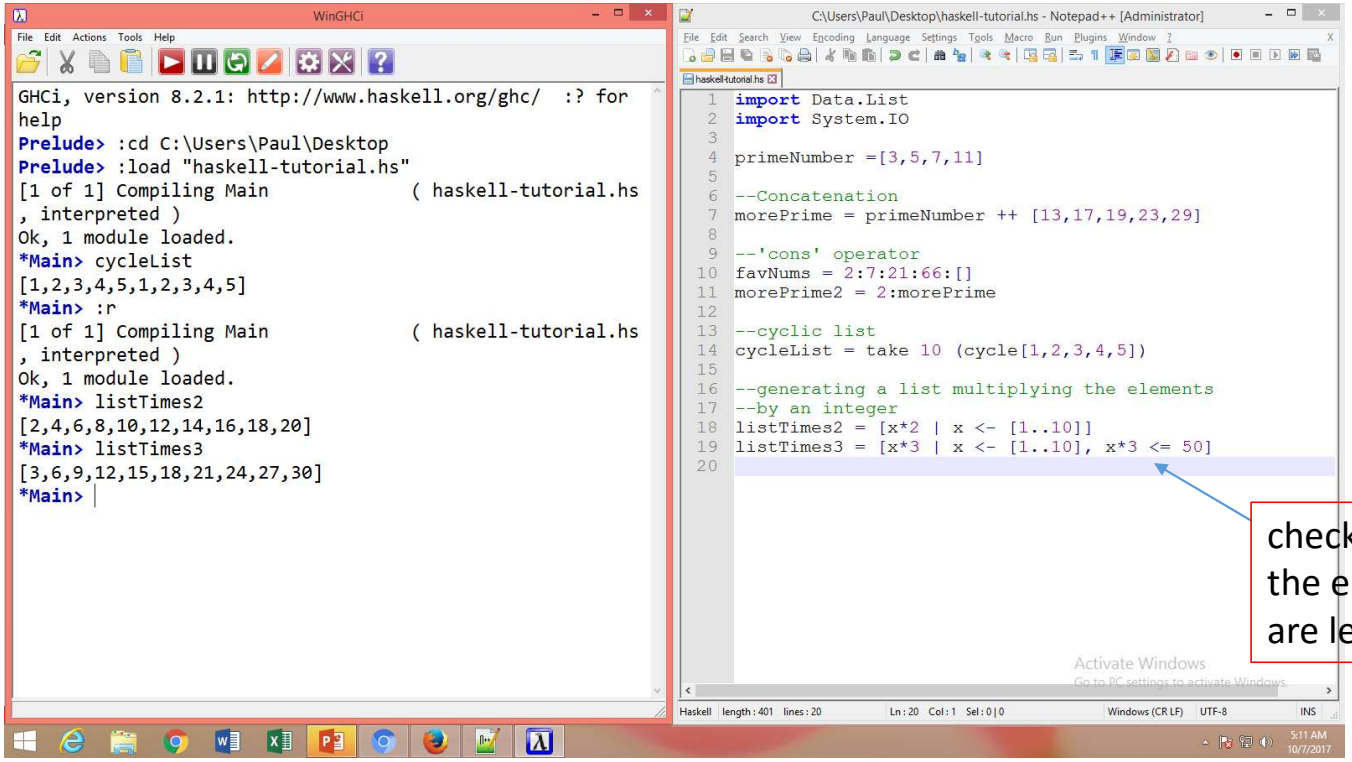


More Operations on List

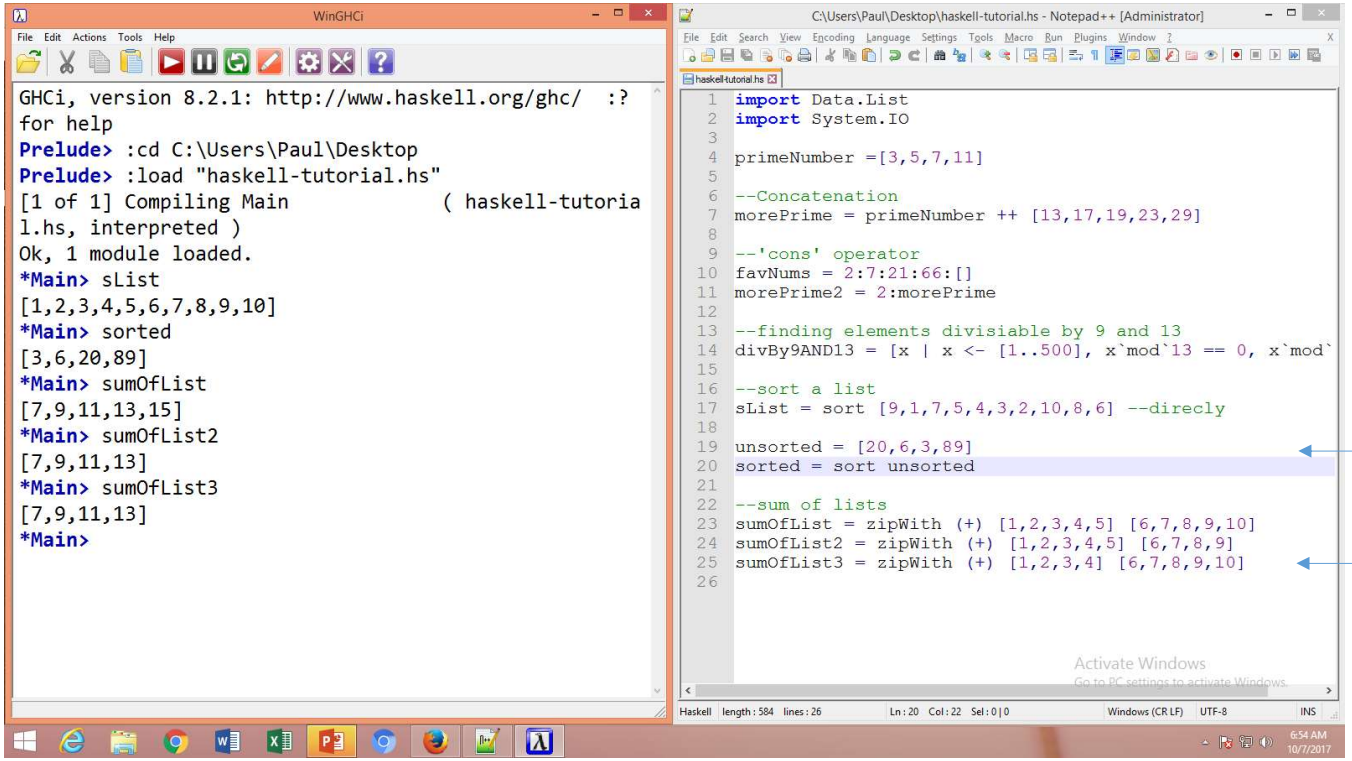


One of the examples of advantages of laziness property and functional approach: here, the presence of ***infinite list*** does not affect other expressions/ functions in the program

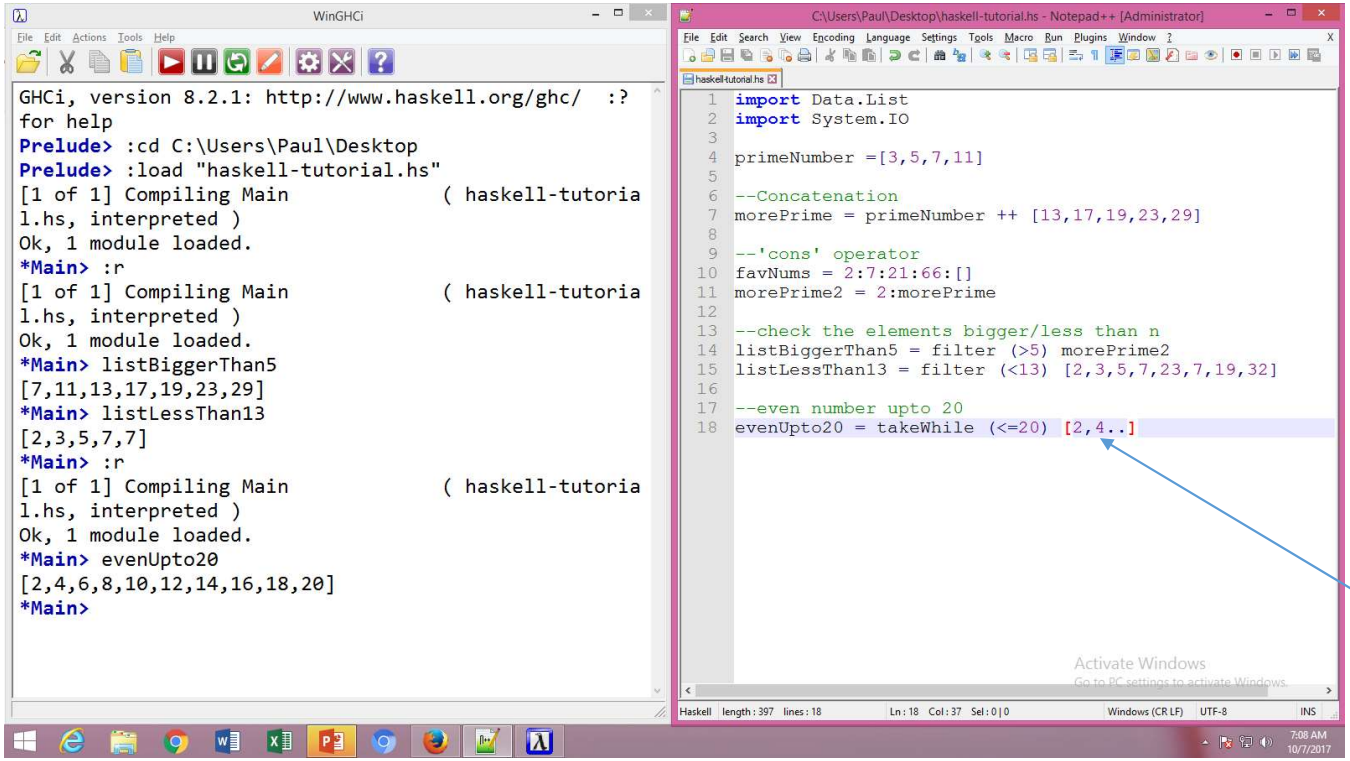
More Operations on List



More Operations on List

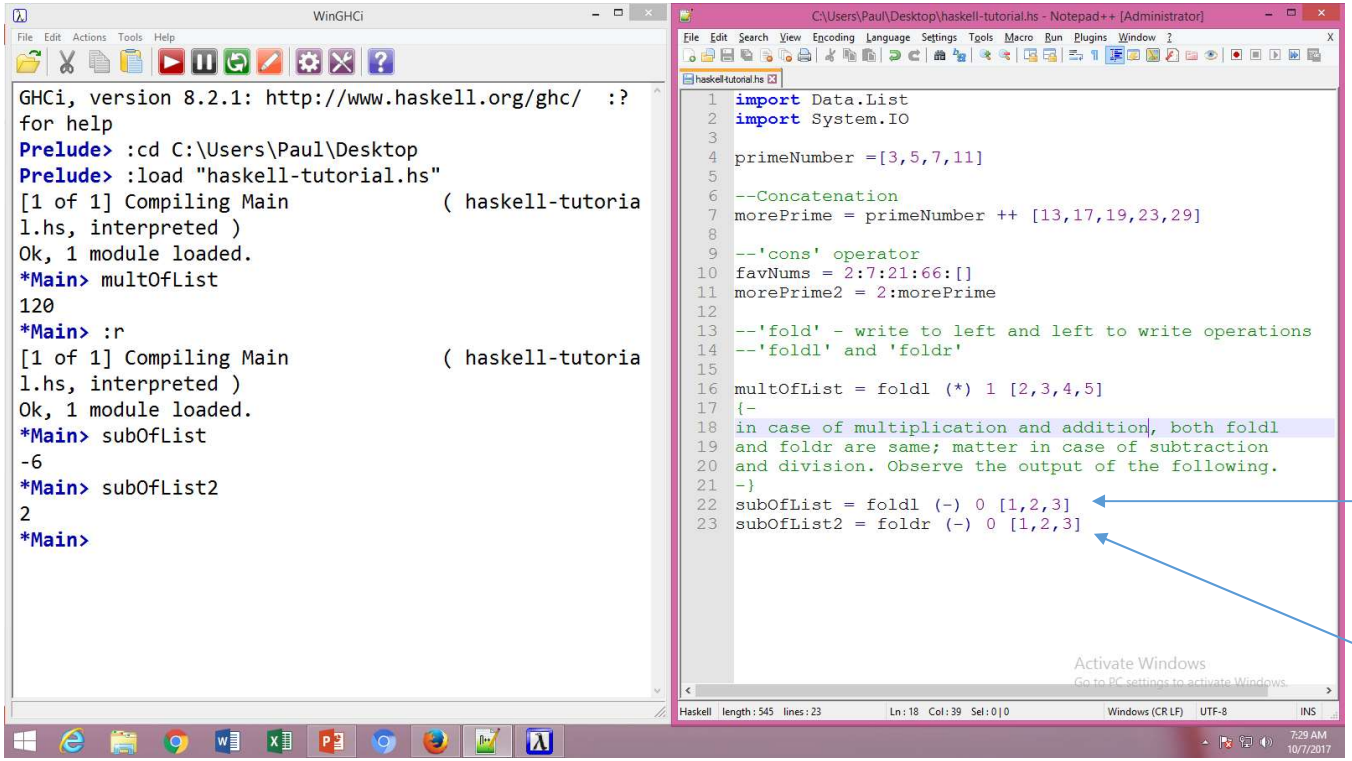


More Operations on List



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

More Operations on List

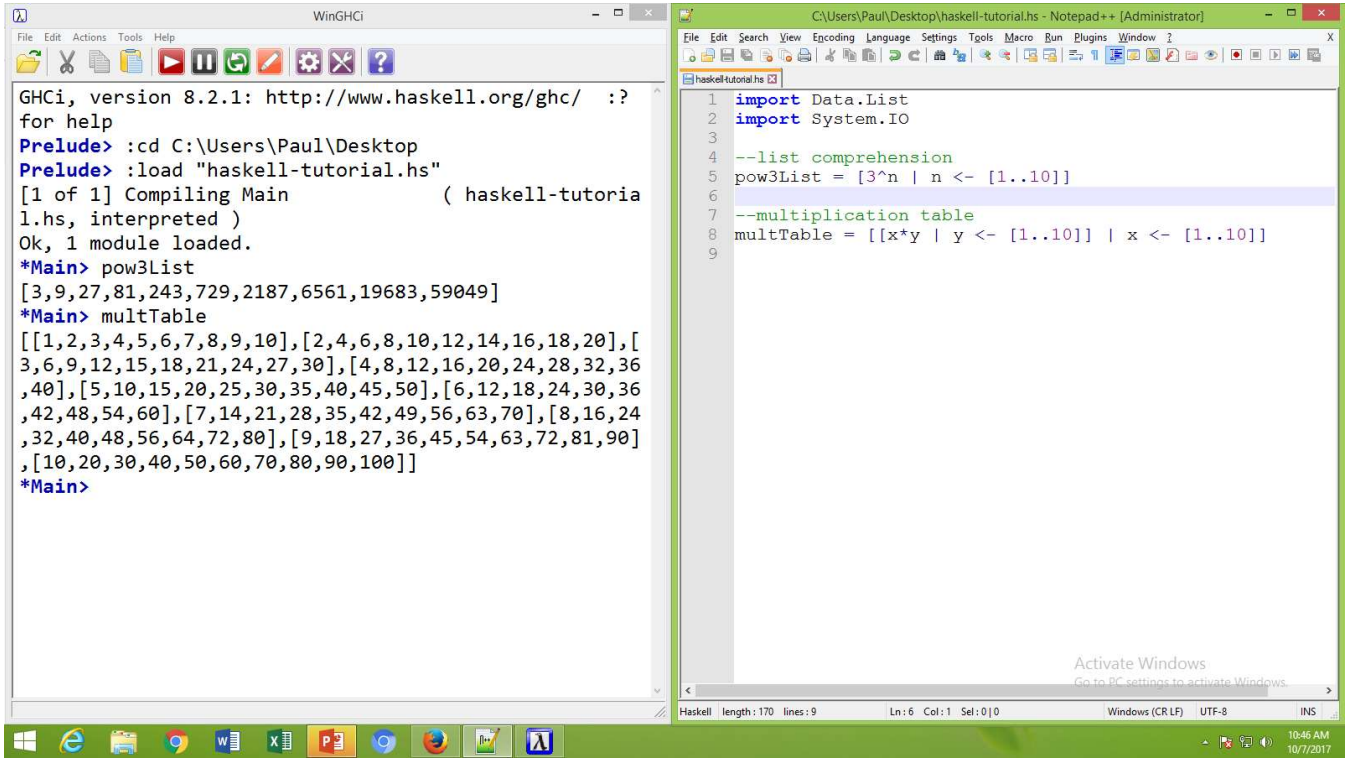


foldl & foldr

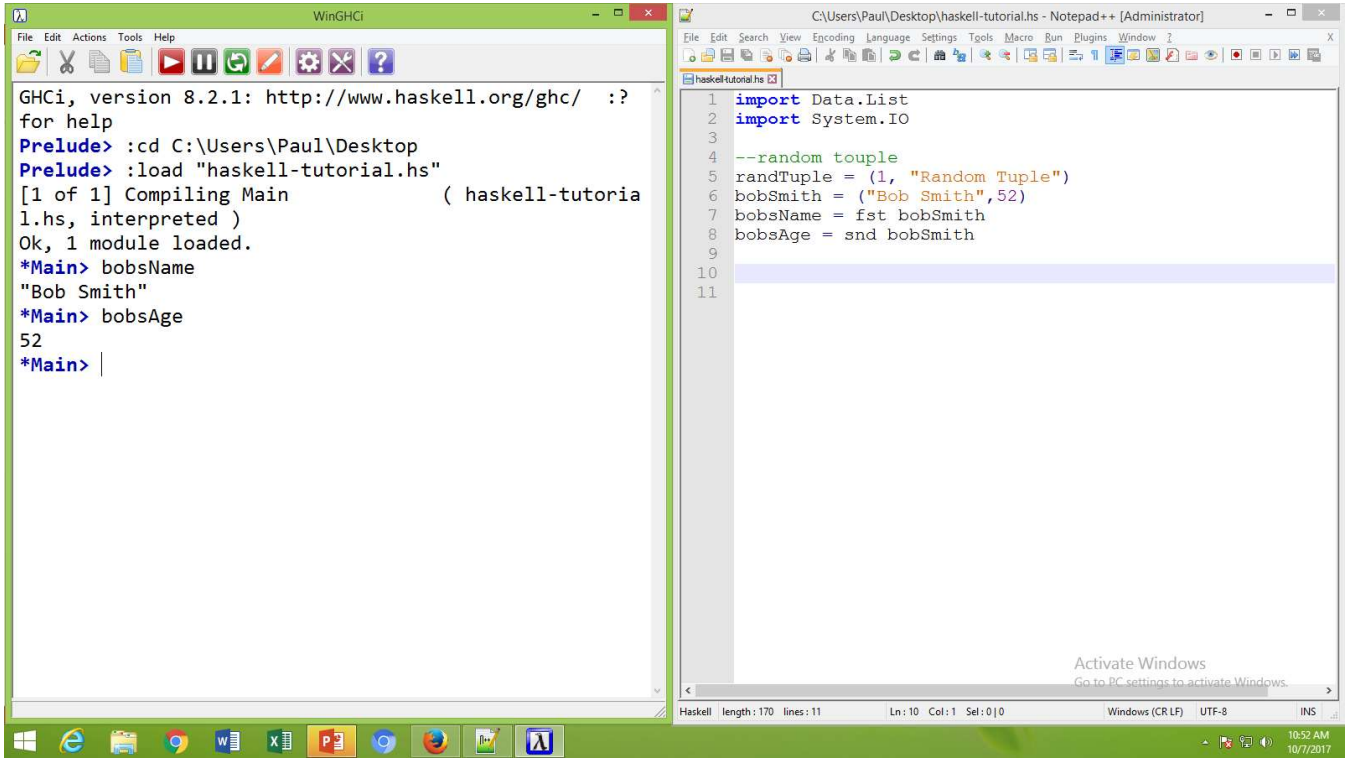
(((0-1)-2)-3)

1-(2-(3-0))

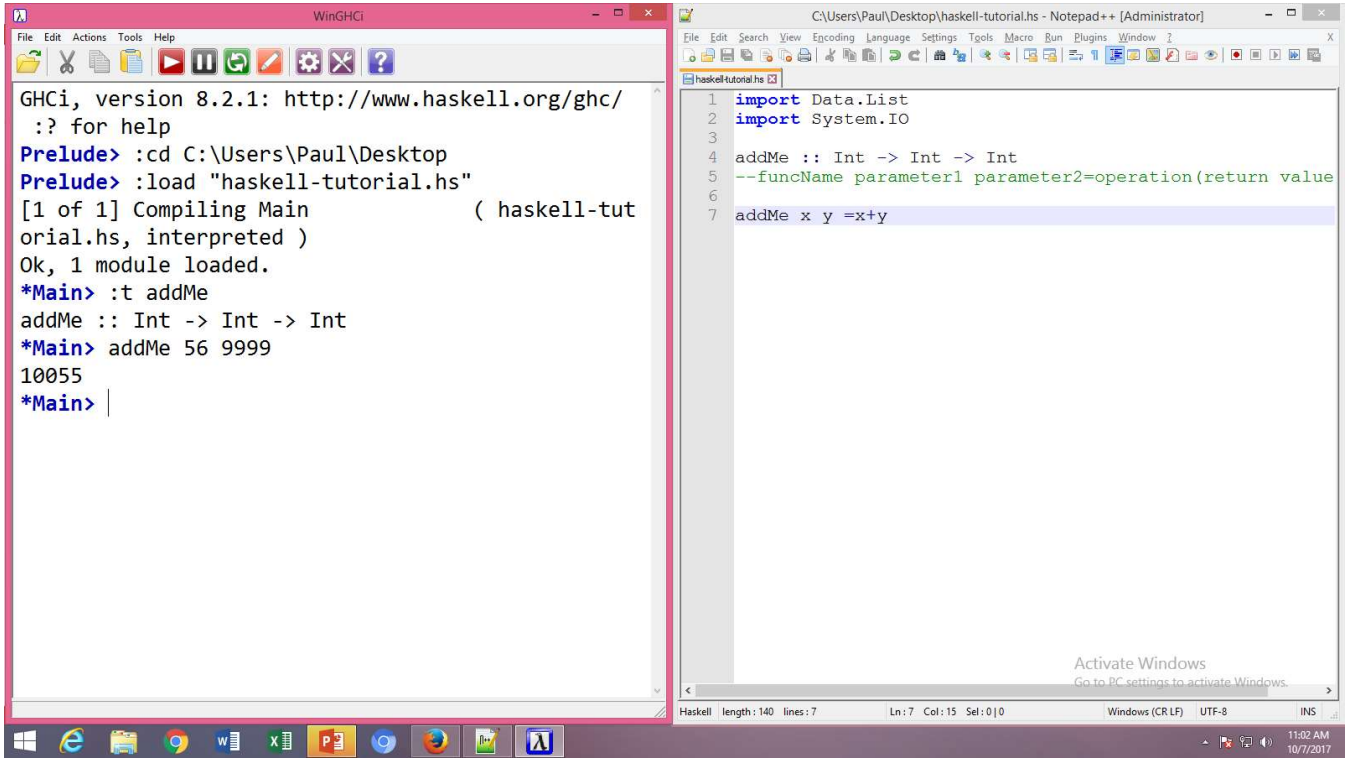
More Operations on List



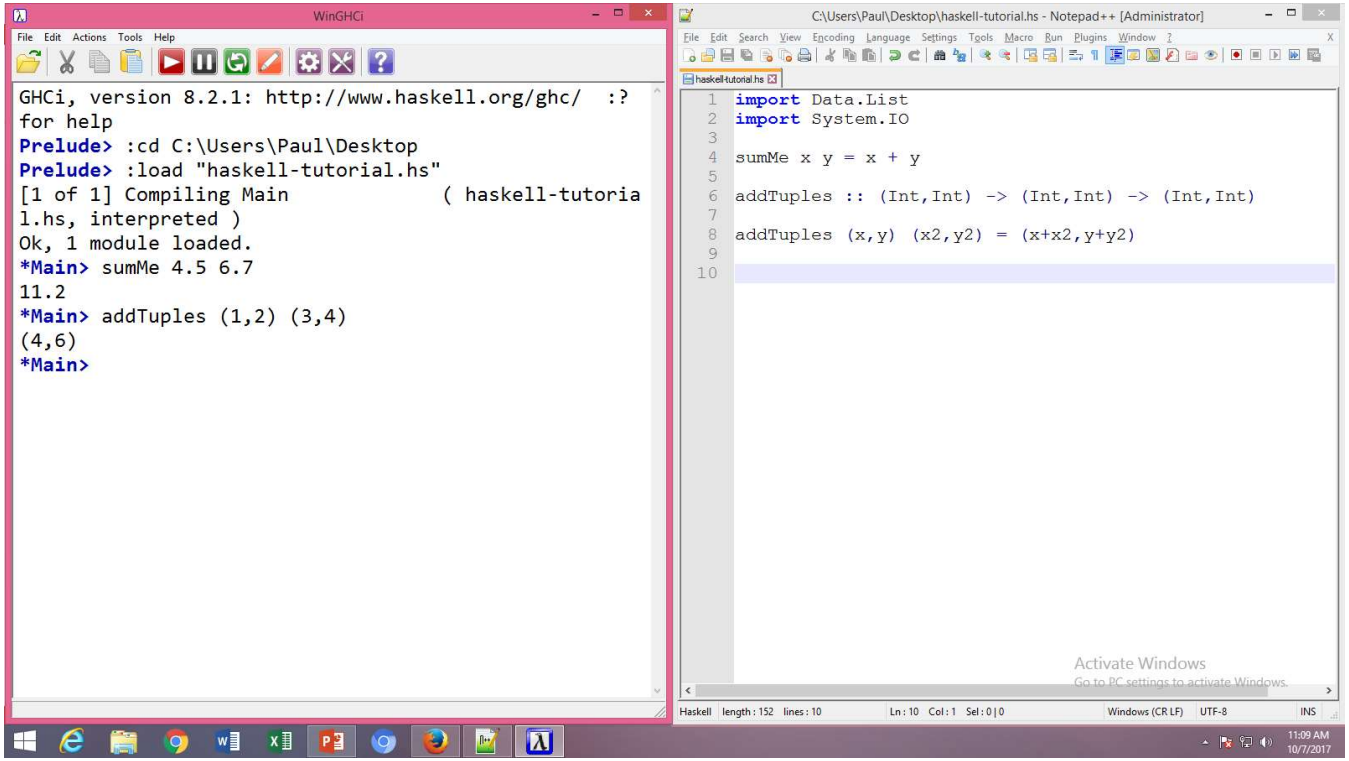
Multiple Data Type



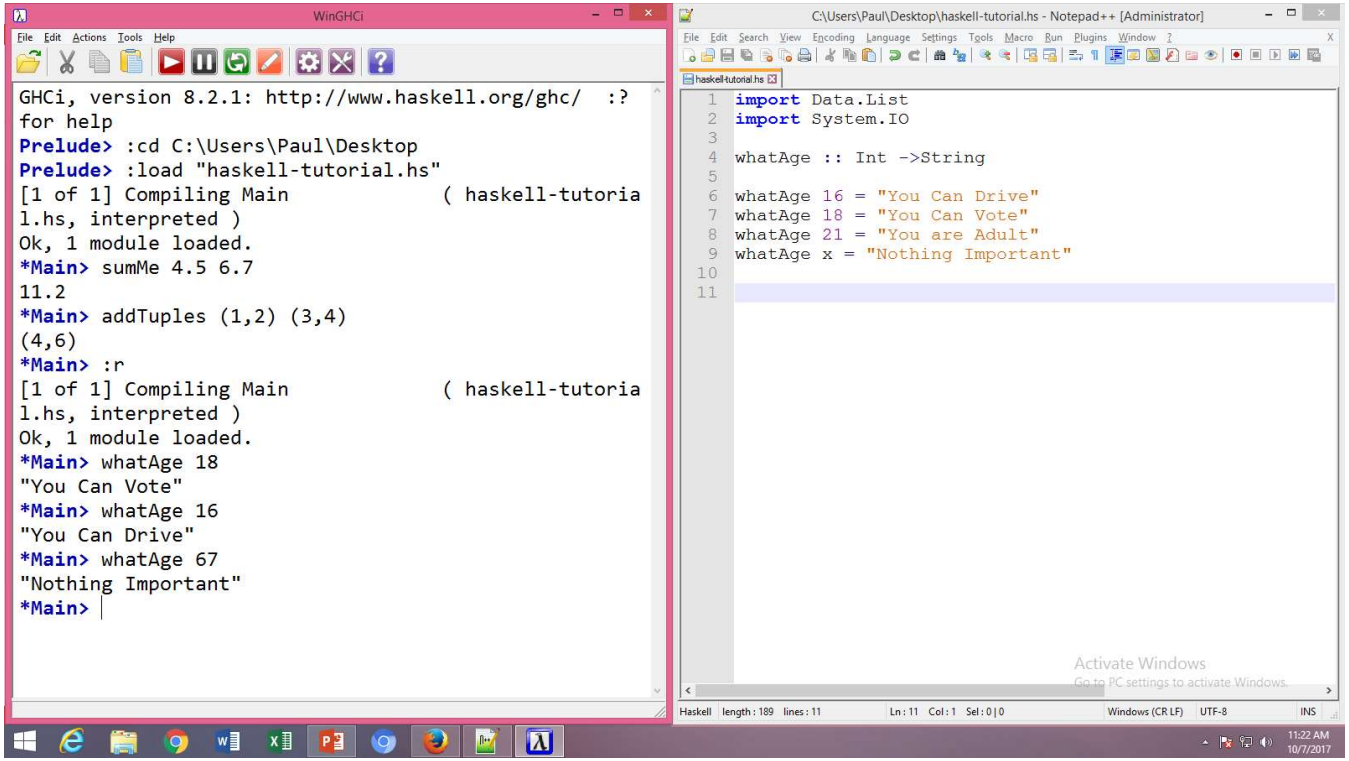
Function Declaration



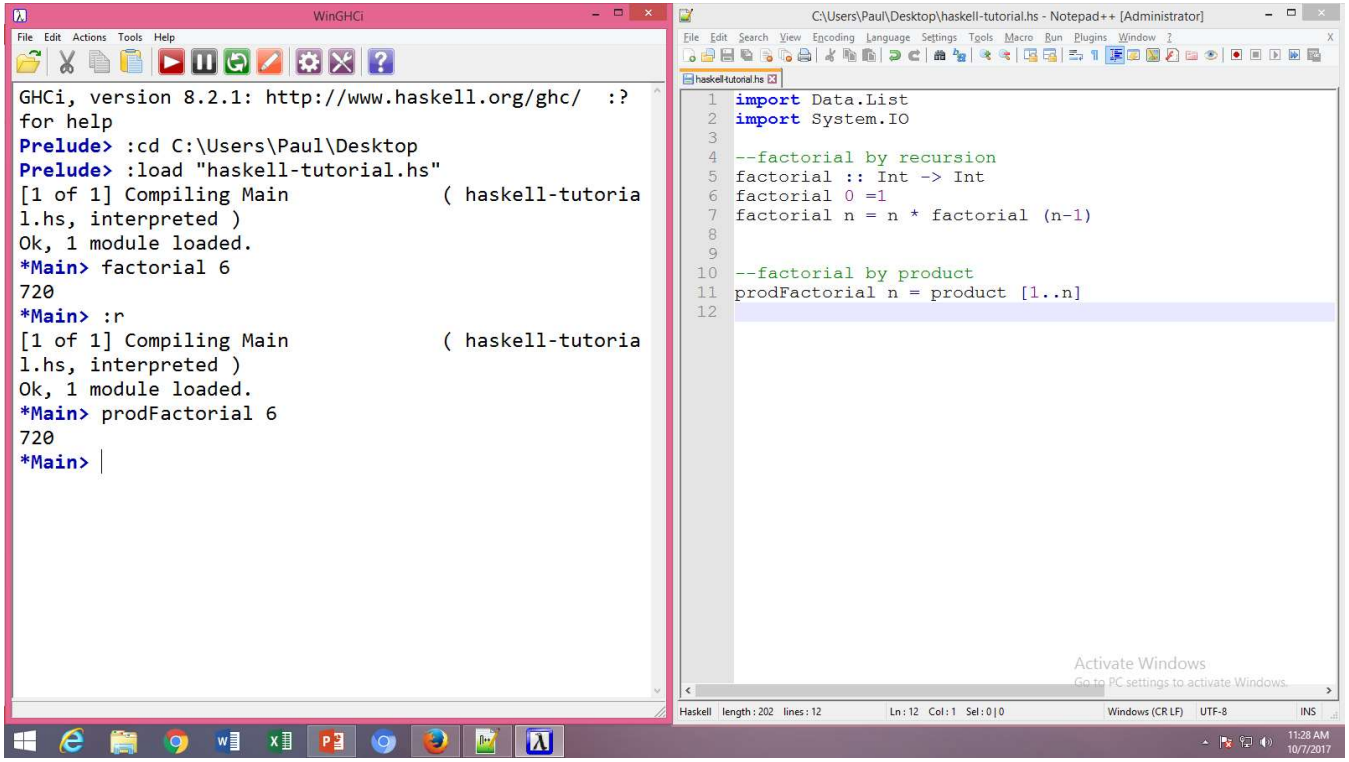
User Type Declaration



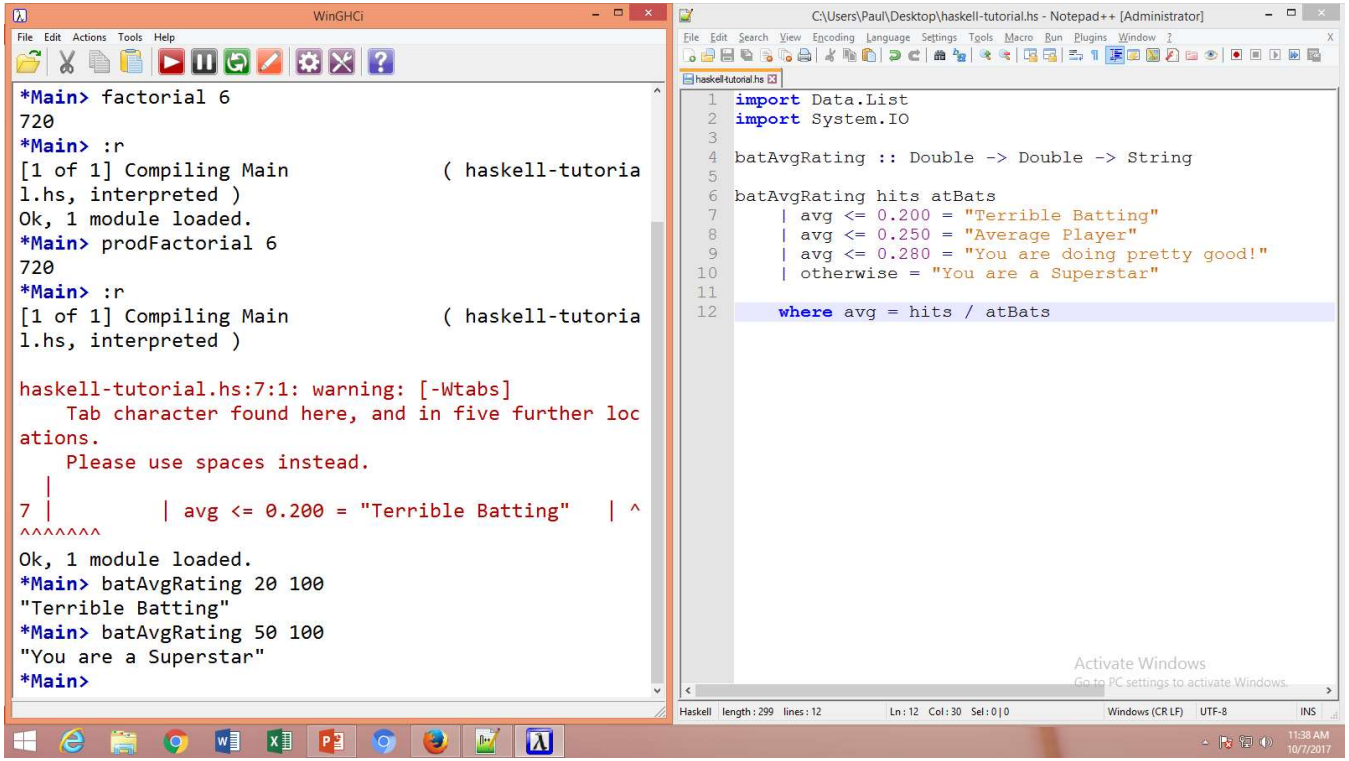
User Type Declaration



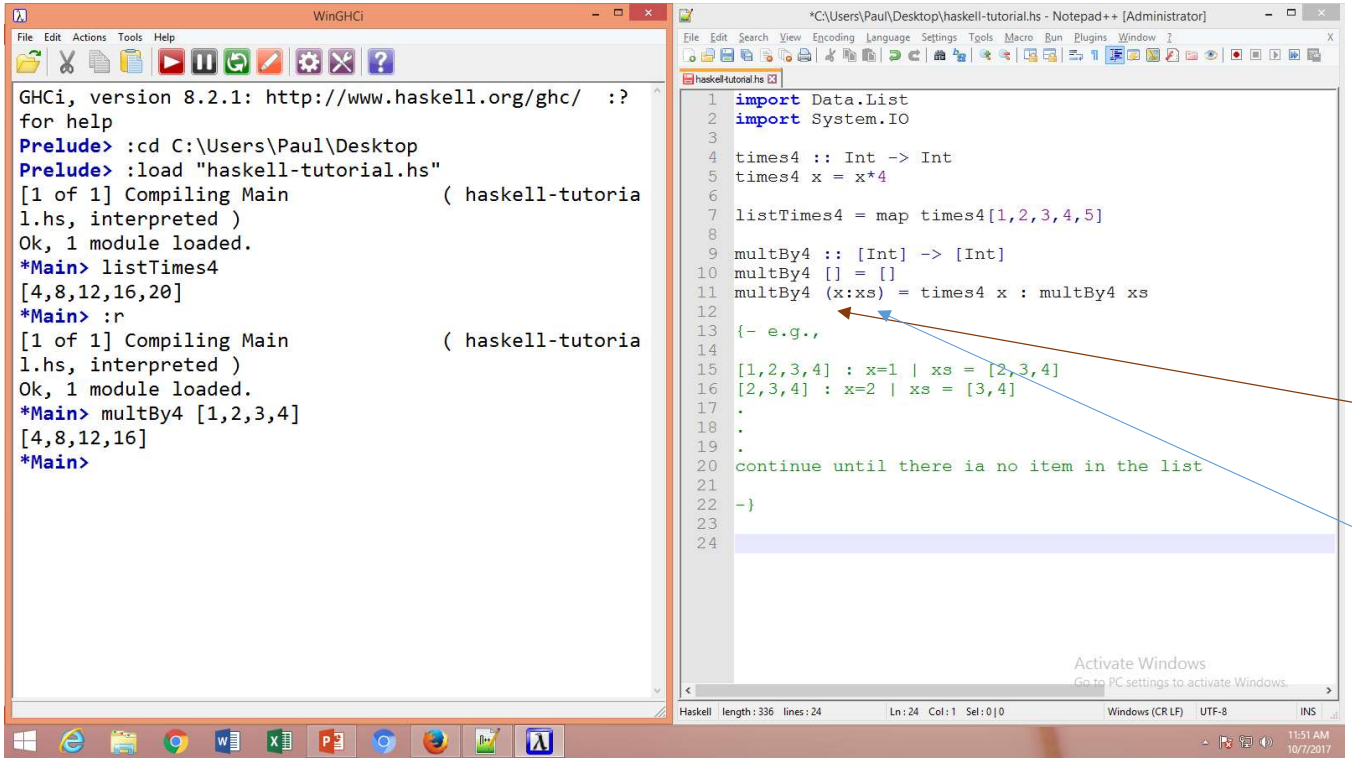
Factorial (by recursion and by product)



Guard (where clause)

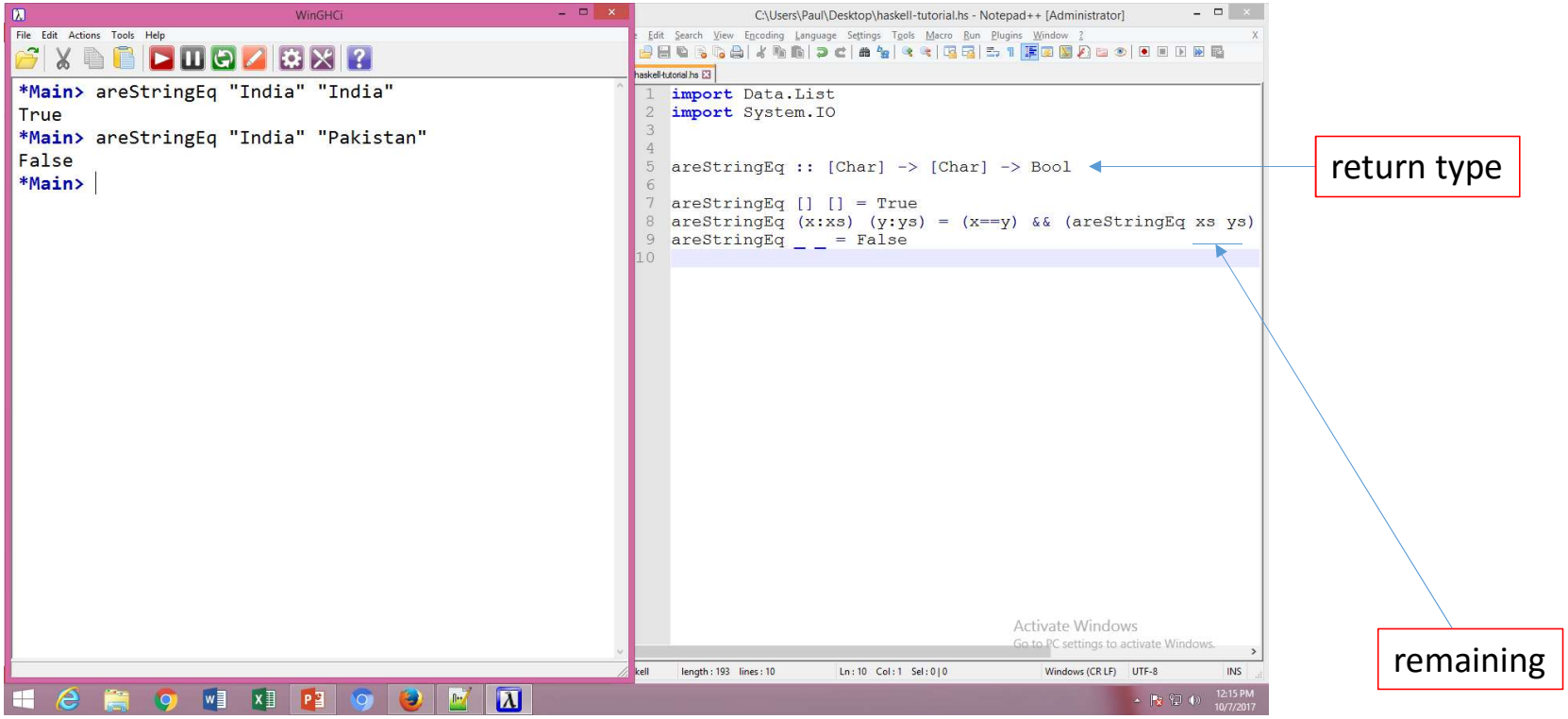


Higher Order Functions

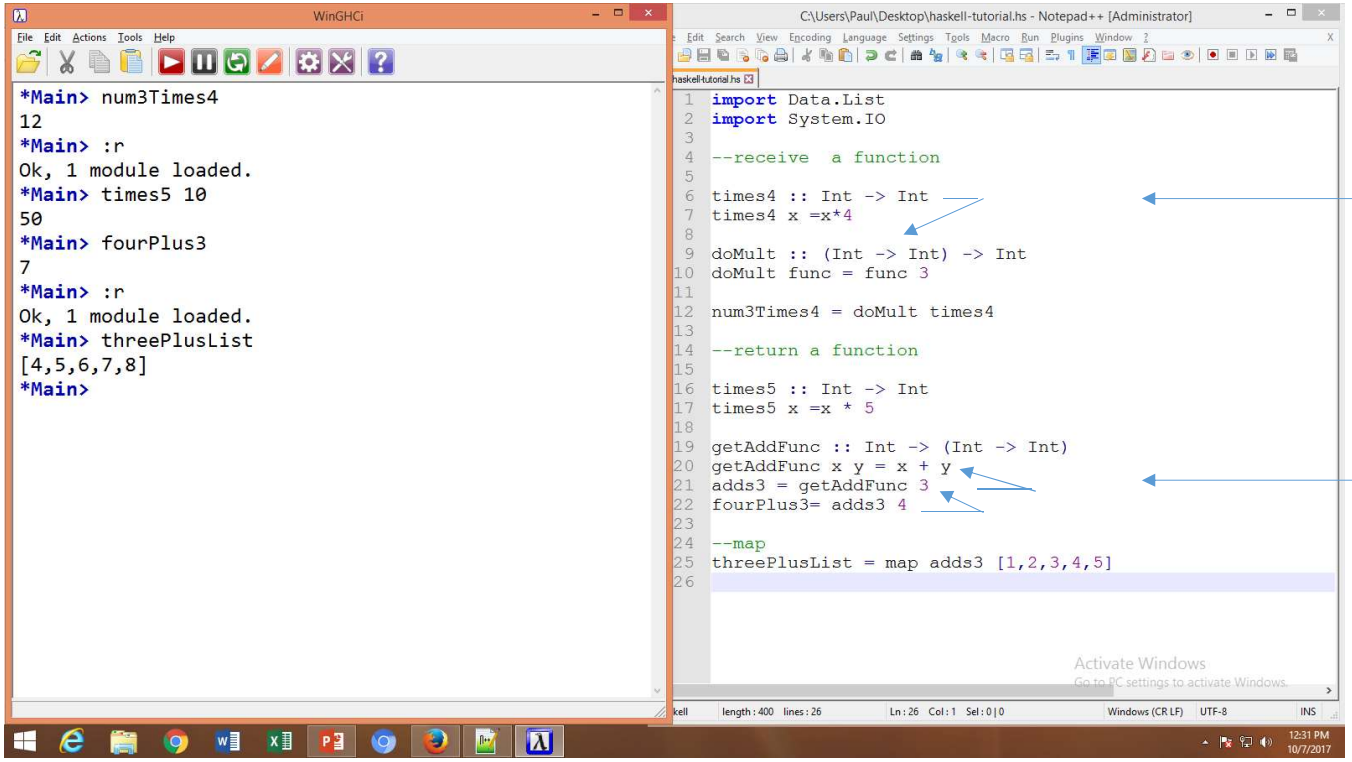


you don't know how many items in the list Beforehand; x represents first element in the list, and xs represents remaining elements of the list

Higher Order Functions



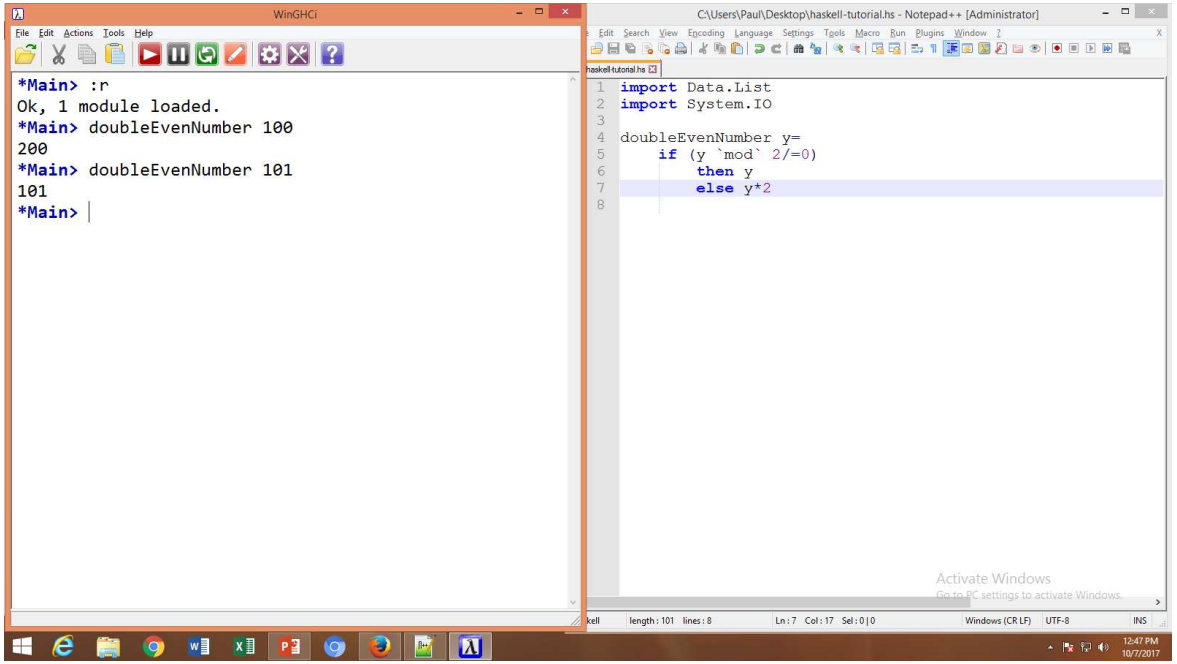
Receive and Return a Function



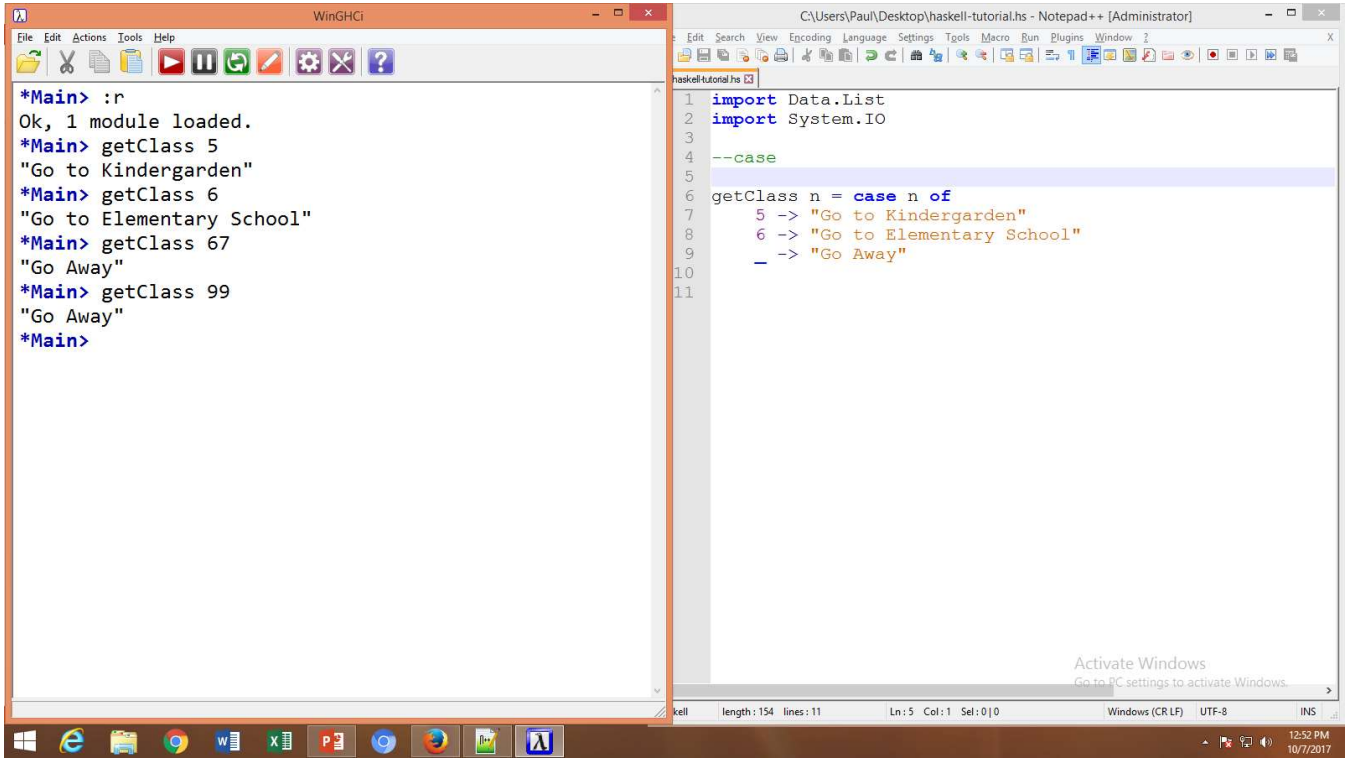
Other Operators

- Comparison
  - `<` --less than
  - `>` --greater than
  - `<=` --less than equal to
  - `>=` --greater than equal to
  - `==` --equal to
- Logical
  - `&&` --AND
  - `||` --OR
  - `not` --NOT

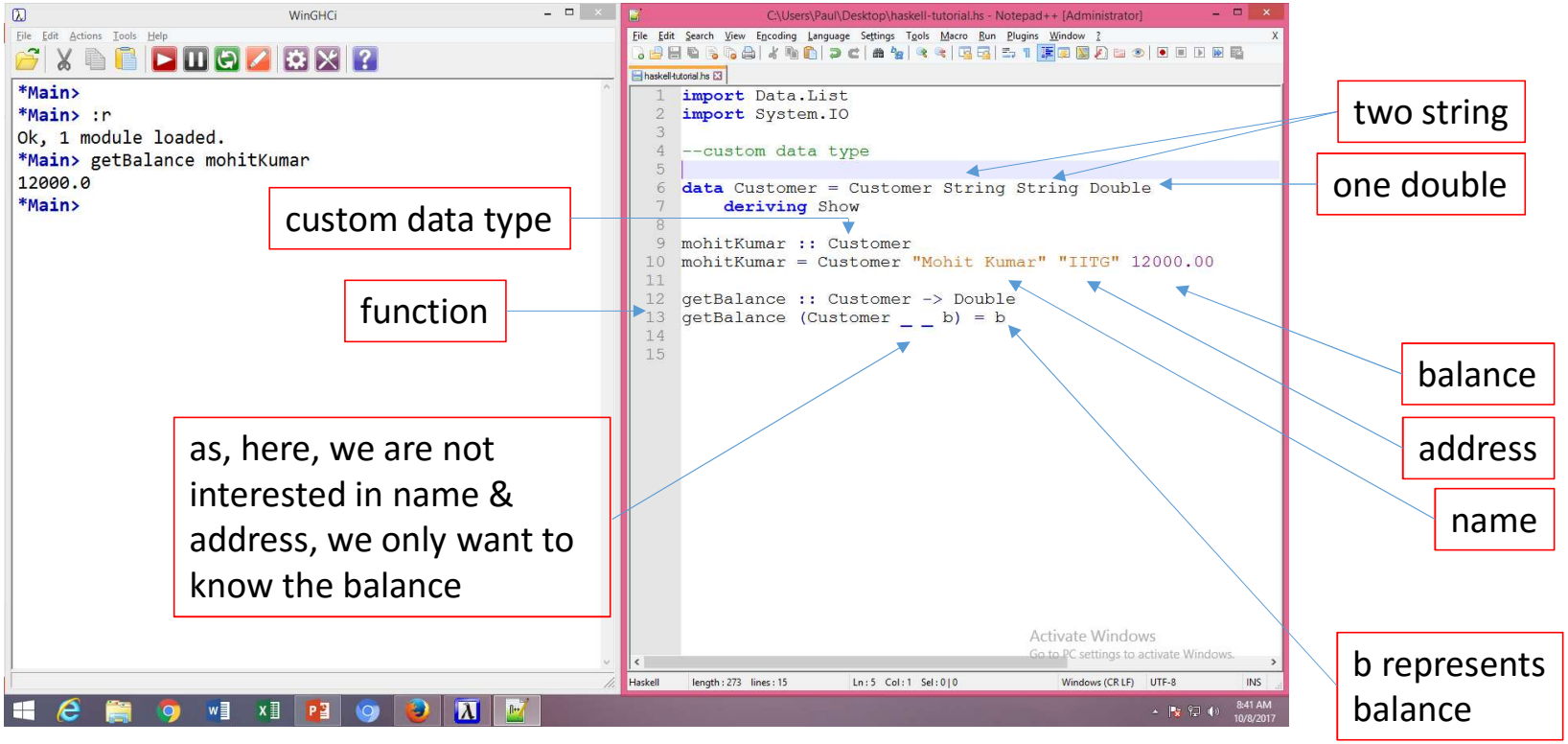
Example



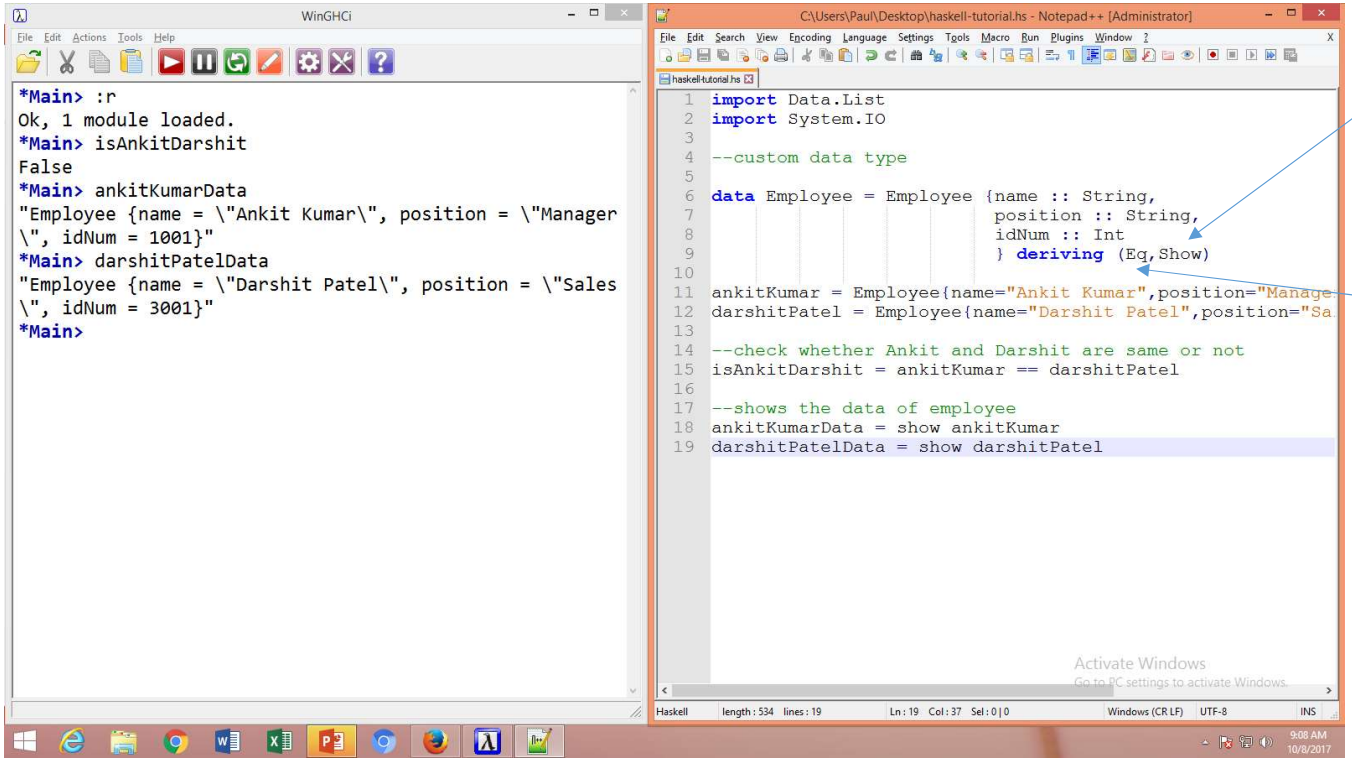
Case


WinGHCi
File Edit Actions Tools Help
*Main> :r
Ok, 1 module loaded.
*Main> getClass 5
"Go to Kindergarden"
*Main> getClass 6
"Go to Elementary School"
*Main> getClass 67
"Go Away"
*Main> getClass 99
"Go Away"
*Main>
C:\Users\Paul\Desktop\haskell-tutorial.hs - Notepad++ [Administrator]
haskell-tutorial.hs
1 import Data.List
2 import System.IO
3
4 --case
5
6 getClass n = case n of
7 5 -> "Go to Kindergarden"
8 6 -> "Go to Elementary School"
9 _ -> "Go Away"
10
11
Activate Windows
Go to PC settings to activate Windows.
length : 154 lines : 11
Ln : 5 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS
12:52 PM
10/7/2017

Custom Data Type



Type Classes



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