**Haskell Assignment 1**

1. What values do you expect Haskell to return for each of the following?
2. **2 + 3**

* **Expected Result: 5**

1. **2 == 3**

* **Expected Result: 2 is 3**

1. **[5] ++ [4]**

* **Expected Result: [5, 4]**

1. **5 : [4]**

* **Expected Result: [5, 4]**

1. **4 / 2**

* **Expected Result: 2**

1. **(/) 4 2**

* **Expected Result: 2**

1. **div 4 2**

* **Expected Result: 2**

1. **4 `div` 2**

* **Expected Result: 2**

1. **(\x -> x + 1) 4**

* **Expected Result: 5**

1. **foldr1 (+) [1,2,3,4]**

* **Expected Result: 10**

1. **map (\x -> 1 + x) [2,3]**

* **Expected Result: [3, 4]**

1. **map (+ 1) [2, 3]**

* **Expected Result: [3, 4]**

1. **filter even [1,2,3,4]**

* **Expected Result: [2, 4]**

1. **filter (\x -> x > 2) [1, 2, 3, 4]**

* **Expected Result: [3, 4]**

1. **[n + 2 | n <- [1 ..3]]**

* **Expected Result: 3, 4, 5**

1. **5 : [b | b <- [1..10], b > 5]**

* **Expected Result: [5, 6, 7, 8, 9, 10]**

1. **[2 \* x | x <- [3, 4, 5, 6], x `mod` 2 == 0]**

* **Expected Result: [3, 4, 5, 6]**

1. See if you were right, for example using **tryhaskell.org**. If you were wrong, briefly explain what you learned from it.

* **b. 2 == 3** is not 2 is 3 because in Haskell that expression is actually referring to a Boolean expression and so it actually says is 2 equal to 3 thus the actual result is false.
* **o. [n + 2 | n <- [1 ..3]]** is not 3,4,5 because since [1 .. 3] is in a list than the expected result/output must also be in a list format and not 3 individual integers.The actual expected result is [3,4,5]
* **q. [2 \* x | x <- [3, 4, 5, 6], x `mod` 2 == 0]** is not [3,4,5,6] because after playing around with the expression I have learned that we must first take the list given and only concern ourselves with integers that have a factorial or can be divided by 2. In this case, this will only leave us with [4,6]. Then we use this new list and follow the remaining rule of 2 \* x to each element/integer in the list thus resulted the actual expected result of [8, 12]

1. do the tutorial at **tryhaskell.org**, through Lesson 4,
2. doChapters 1 and 2 of the tutorial at **Y**,
3. doChapters 3 and 4 of the tutorial at **learnyouahaskell.com**, skipping typeclases, case expressions, and $,
4. doChapter 5 of the tutorial at **learnyouahaskell.com**, unless you’re already very skilled with recursion, and don’t feel the need to review.