CS360

Lab one

Programming Languages: Haskell

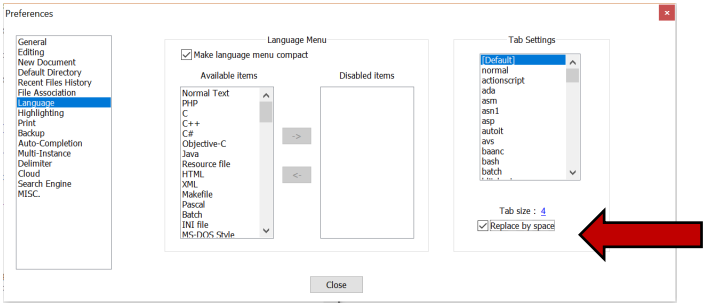
Learn You a Haskell for Great Good

Get yourself set up with a Haskell installation. You need to use The Haskell Platform. It's big but nice and easy and comes with everything you need.

1. All Haskell labs must have a main as follows:

```
main=do
        print (squareRoot 818281336460929553769504384519009121840452831049)
```

   a. Every function must have a print (*nameOfFunction parameters)*
   b. The main=do is at the scope furthest out
   c. Each print must be indented two tabs
   d. Your tabs must be set to spaces.
      (Settings ->Preferences-> *as shown below* ***check Replace by space***



2. **IMPORTANT! – When I click on your .hs file it will run the main automatically. Labs failing to run from the main will receive a zero.**
3. **While you are reading**, have a session in GHCI open. **For each and every concept you read about**, try it out. This means typing in many examples from the book **and** trying out your own versions. Just to confirm, yes type it yourself, do not copy and paste! You'll store it better in your brain if you're the one typing it. And when you are trying out your own versions, try to think of other things to try. Be inquisitive. For example, the book says that GHCI will yell at you for trying to evaluate `5 * -3`. But `-3 * 5` works just fine. Why? Not all your examples should work, that's part of learning the syntax.

For each of the following, ***write a Haskell expression that answers the question and then include the answer***. Your code should look like this ->

```
doubleMe x = x + x
-- passing in 5 evaluates to 10
```

1. What is the square root of `818281336460929553769504384519009121840452831049`?

2. What is the ASCII character that comes before uppercase `A`?
3. What is a function that evaluates to True or False, that tells you whether or not three times a number plus one is even? Try it out on a couple numbers.
4. What is the product of all the odd numbers one to one hundred?
5. Use list operation functions like `head`, `tail`, last, etc. to find the greatest number in the following list, that is not the first or last number. i.e. if the first and last numbers don't count, then what is the largest number in the "middle" of the list?

```
[99,23,4,2,67,82,49,-40]
```

6. Using the list construction operator, `:`, and the empty list, `[]`, write an expression to construct a list like this: `[6,19,41,-3]`, i.e. don't just write the list, *construct it*.

Write Haskell expressions that use list comprehensions to solve the following problems.

7. What are the first 27 even numbers? Use the `even` function and be Lazy!
8. Provide a list of all odd numbers less than 200 that are divisible by 3 *and* 7.
9. How many odd numbers are there between 100 and 200 and that are divisible by 9?
10. Count how many negative numbers there are in a list of numbers. Input [-4, 6, 7, 8, -14]
11. Use `zip`, Texas ranges and concatenation, to create a list of Hexadecimal mappings like this:

```
[(0,'0'),(1,'1'),(2,'2'),(3,'3'),(4,'4'),(5,'5'),(6,'6'),(7,'7'),(8,'8'),(9,'9'),
(10,'A'),(11,'B'),(12,'C'),(13,'D'),(14,'E'),(15,'F')]
```

12. Write a function that uses list comprehensions to generate a list of lists that works like this:

```
makeList 3 == [[1],[1,2],[1,2,3]]
makeList 5 == [[1],[1,2],[1,2,3],[1,2,3,4],[1,2,3,4,5]]
makeList (-2) == []
```

13. URL's are not allowed to contain spaces. Write a Haskell function that takes a string, and replaces any space characters with the special code %20. For example:

```
sanitize "http://wou.edu/my homepage/I love spaces.html" ==
"http://wou.edu/my%20homepage/I%20love%20spaces.html"
```

You may need the `concat` function, that concatenates a list of lists into a plain list.

14. **Choose 5 of the following functions**, look up their type signature with :t in ghci. Write it down. Then look at the type class hierarchy found on Moodle and write down a few concrete types that it will work with. Include this, commented out, in your .hs file.

    `(*)`

    `(++)`

    `min`

    `length`

    `take`

    `null`

    `head`

    `sqrt`

    `(>)`

    `succ`

    `div`

For the following, implement using *pattern matching of parameters*

15. A function that associates the integers 0 through 3 with "Heart", "Diamond", "Spade", and "Club", respectively. For any other integer values it is an error.

```
getSuit :: Int -> String
```

16. A function that computes the dot product of two 3D vectors (tuples). Follow the Algebraic Definition.

```
dotProduct :: (Double,Double,Double) -> (Double,Double,Double) -> Double
-- e.g.
dotProduct (1,2,3.0) (4.0,5,6) == 32.0
```

17. A function that reverses the order of the first three elements in a list. If the list has fewer than 3 items then reverse the first two, or if it only has one element or is empty, leave it as is. If it has more then 3 elements then reverse the first 3 as asked and leave the remainder of the list as it is.

```
reverseFirstThree :: [a] -> [a]
```

For the following, implement using *guards*

18. Write a function that will tell you how warm or cold it feels like outside given a temperature in Fahrenheit. You can make up the ranges (at least 4) but if you call it with a temperature of (-45.3) it should tell you it is "frostbite central!".

```
feelsLike :: Double -> String
-- e.g.
feelsLike 45 == "Cool"
```

```
feelsLike (-45.3) == "Frostbite Central!"
```

19. Modify your code for the previous problem to convert from celsius to fahrenheit at the same time you're telling what it feels like. Have the user pass in values in celsius this time and you return the temperature in fahrenheit as well as the classification (same as before but after the conversion remember), together in a tuple. Use a *where clause* to do the temperature conversion.

```
feelsLike2 :: Double -> (Double,String)
```

```
e.g.
```

```
feelsLike2 100 == (212.0,"Oven-like")
```

```
feelsLike2 (-1) == (30.2,"Freezing")
```

20. Write a function that uses a let expression and pattern matching of tuples in a list comprehension to calculate the volume of a list of cylinders. You'll take in a list of tuples (where the first value in each tuple is the radius and the second is the height) and return a list of volumes as Double's. Don't use a where clause.

```
cylinderToVolume :: [(Double,Double)] -> [Double]
-- e.g.
cylinderToVolume [(2,5.3),(4.2,9),(1,1),(100.3,94)] ==
[66.6017642561036,498.7592496839155,3.141592653589793,2970842.2548145014]
```