Functional Programming in Haskell

Prof. Madhavan Mukund & S.P Suresh

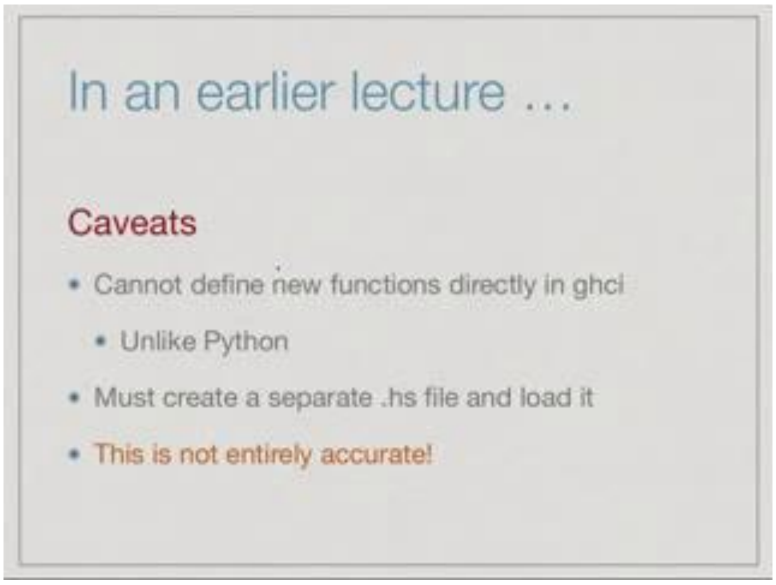
Chennai Mathematical Institute

Module # 04

Lecture – 05

Defining functions within ghci

(Refer Slide Time: 00:03)



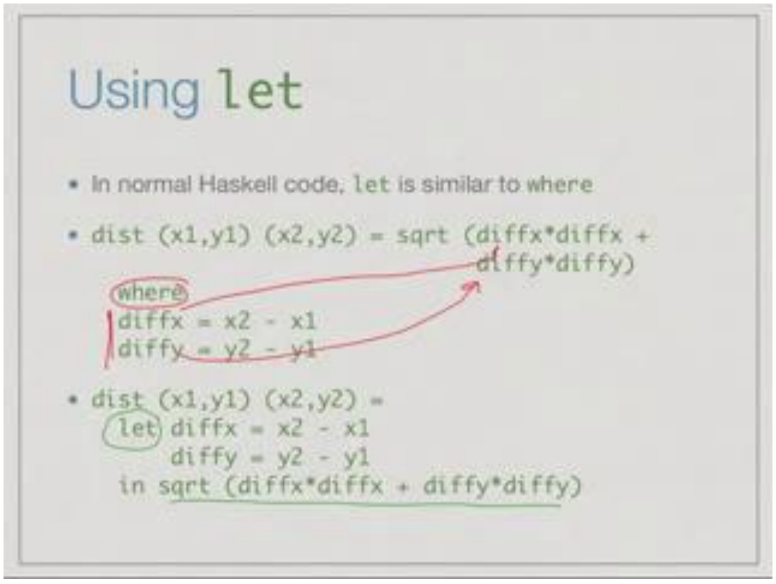
In an earlier lecture ...

Caveats

- Cannot define new functions directly in ghci
  - Unlike Python
- Must create a separate .hs file and load it
- This is not entirely accurate!

So, in an earlier lecture in the first week, we introduce the interpreter ghci. We said that we cannot define new functions directly in ghci, unlike say in python and you must create a separate Haskell file with extension dot hs and load it into ghci. We turns out that this is not entirely accurate.

(Refer Slide Time: 00:24)



Using let

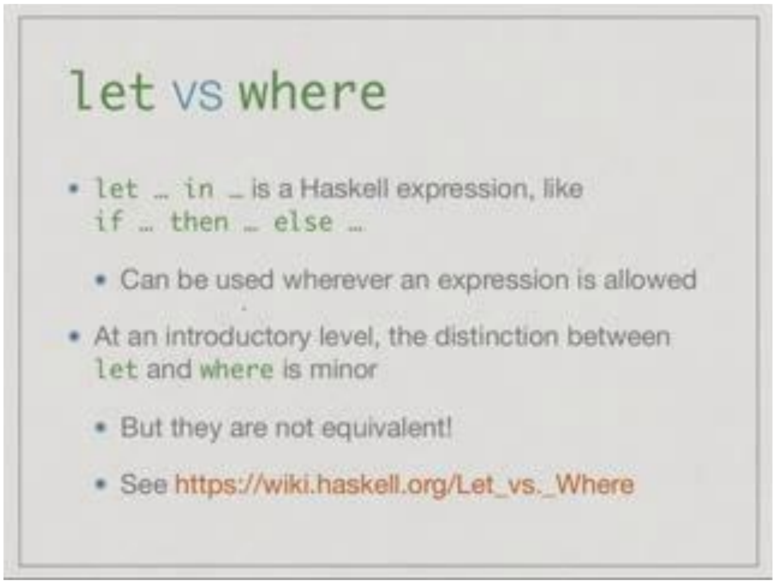
- In normal Haskell code, let is similar to where

-
  ```
  dist (x1,y1) (x2,y2) = sqrt (diffx*diffx +
                               diffy*diffy)
    where
    diffx = x2 - x1
    diffy = y2 - y1
  ```

-
  ```
  dist (x1,y1) (x2,y2) =
    let diffx = x2 - x1
        diffy = y2 - y1
    in sqrt (diffx*diffx + diffy*diffy)
  ```

So, in order to use functions, definitions in ghci we need to learn a little bit more about Haskell. So, we have seen the use of the word where in order to specify local definitions and functions. So, we can define a function in terms of some local definition and use these definitions in the function. An equivalent way of doing this is to use an expression called let. So, we can instead of using where, we can say let diff x be x 2 minus x 1, diff y be y 2 minus y 1 in this definition. So, these are seemingly two equivalent ways to write it, we have looked at where so far, we have not seen let.

(Refer Slide Time: 01:10)



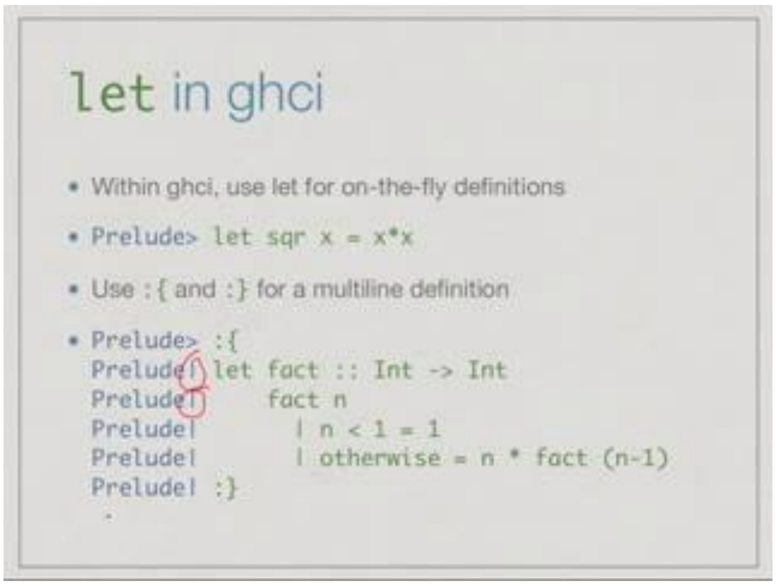
let vs where

- let … in … is a Haskell expression, like if … then … else …
  - Can be used wherever an expression is allowed
- At an introductory level, the distinction between let and where is minor
  - But they are not equivalent!
  - See https://wiki.haskell.org/Let_vs._Where

So, let something, in something is a Haskell expression, is like if something, then something, else something which is also Haskell expression. It can be used wherever an expression is allowed, at a level it which we are using Haskell, there is no significant difference between let and where, which is why we are used where so far. The distinction is minor for us, but actually because let is a full fledged Haskell expression and where is not.

They are not equivalent in more complicated context as we may see when we go along. If you are curious, you can look up this link to find out some ways in which let differs somewhere.

(Refer Slide Time: 01:49)



let in ghci

- Within ghci, use let for on-the-fly definitions
- Prelude> let sqr x = x*x
- Use :{ and :} for a multiline definition

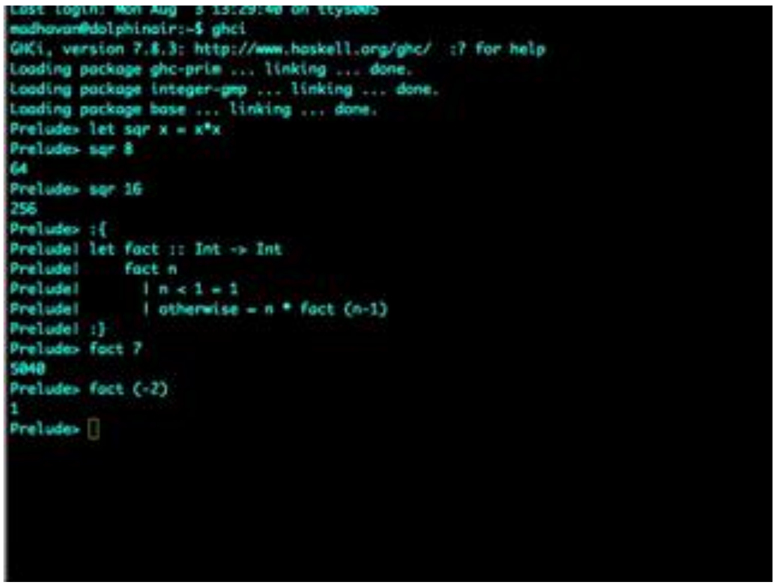
-
  ```
  Prelude> :{
  Prelude| let fact :: Int -> Int
  Prelude|     fact n
  Prelude|       | n < 1 = 1
  Prelude|       | otherwise = n * fact (n-1)
  Prelude| :}
  ```

And this moment what is relevant for us is, that we can use let inside ghci to define functions on the fly. At a basic level we can use let to define single line values, so we can write something like let square of x equal to x square x and now square will be available within ghci. So, therefore, it is definitely possible to actually define functions indirectly in ghci and not exactly the way you do it outside in a Haskell file, but we are using let.

Of course, this is a single line definition. What if we want is a multiple line definition, like say factorial which is defined separately for 0 and 1. So, we can use this notation colon open brace and colon close brace to enclose a multiple line definition. So, if we say colon open brace, notice that you will find a slide change in the path ghci which is greater than in, will give you something else perhaps a type, symbol like this.

Until you put a close brace and then the next line will put term back to the usual form and in between, you can write a multiple line that let for example, we can say something like, let fact int to int. Be the function your fact of n, if n is less than 1 is 1 otherwise, it is n times fact n minus 1.

(Refer Slide Time: 03:08)

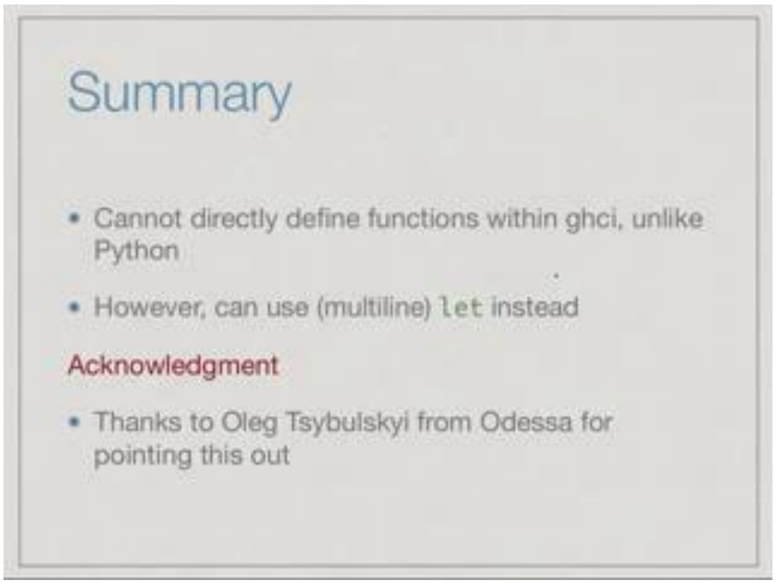


```
Last login: Mon Aug  3 13:29:40 on ttys005
madhavan@dolphinair:~$ ghci
GHCi, version 7.8.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> let sqr x = x*x
Prelude> sqr 8
64
Prelude> sqr 16
256
Prelude> :{
Prelude| let fact :: Int -> Int
Prelude|     fact n
Prelude|       | n < 1 = 1
Prelude|       | otherwise = n * fact (n-1)
Prelude| :}
Prelude> fact 7
5040
Prelude> fact (-2)
1
Prelude> 
```

So, here let us actually do this to verify that this words. So, we say ghci, then we can say let square x equal to x star x or we x square of 8 for instance, we get 64. We say square of 16 we get 256 and so on. Now, we want a multi line definition, we can think prompt changes. Now, we can say let fact int to int be given by fact n, such that n is less than 1 is equal to 1 otherwise, it is equal to n times fact of n minus 1 and then, we close this multi line definition, we get back to whole funct.

Now, we say, what is factorial of 7, to get 5040, we say what is factorial of minus 2, we get 1 and so on. So, we can use let with the open brace close brace if necessary to define functions in ghci indirectly, not exactly the same way we do it in the Haskell file if we source through the load function. But, effectively we can write functions on the fly.

(Refer Slide Time: 04:21)



Summary

- Cannot directly define functions within ghci, unlike Python
- However, can use (multiline) `let` instead

Acknowledgment

- Thanks to Oleg Tsybulskyi from Odessa for pointing this out

So, to summarize we cannot directly define functions within ghci unlike python, in python this same def command which is used to define functions in a file, is exactly what we use in the interpreter. In Haskell, you have to use let perhaps with this multi line colon open brace, colon close brace and I would like to thank Oleg Tsybulskyi from Odessa for pointing this out on the discussion for us.