Introduction To Haskell Programming

Prof. S.P Suresh

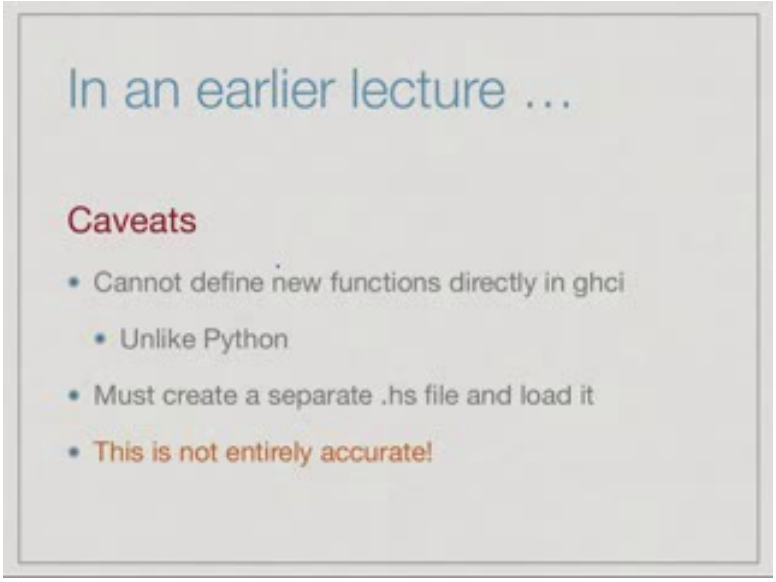
Chennai Mathematical Institute

Module # 04

Lecture – 05

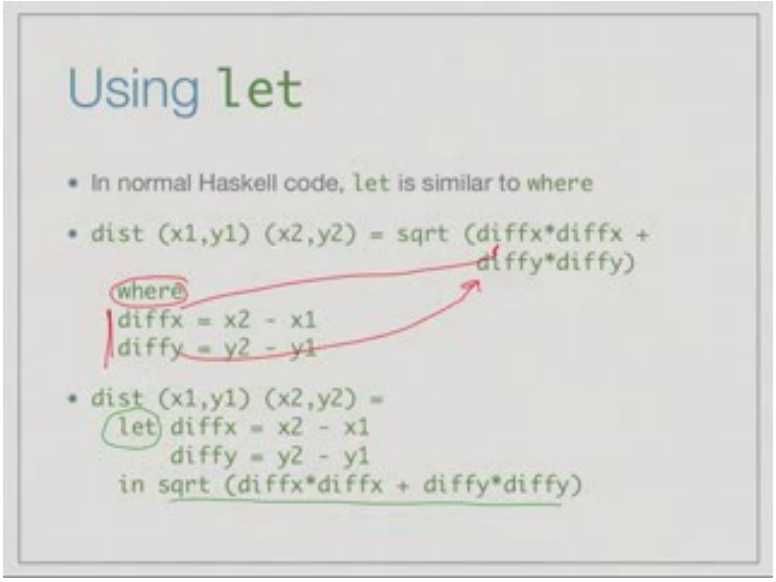
Defining functions within ghci

(Refer Slide Time: 00:03)



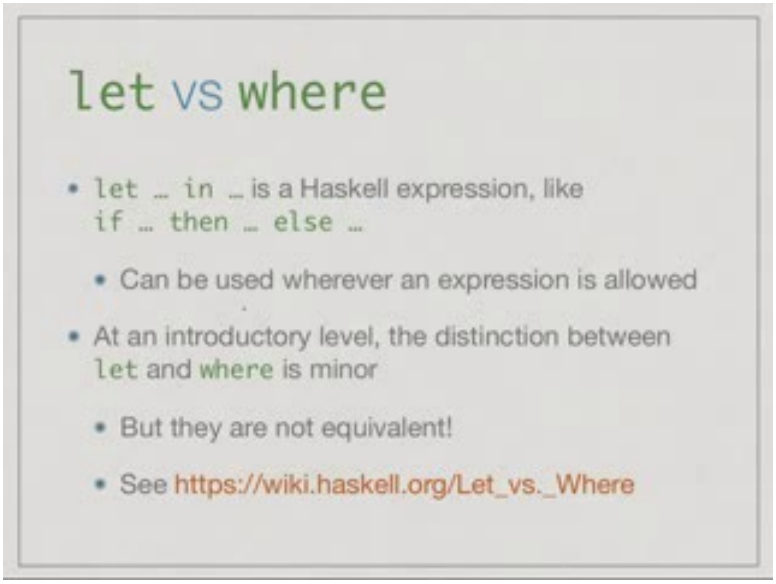
So, in an earlier lecture in the first week when we introduced the interpreter ghci, we said that you cannot define new functions directly in ghci, unlike say in python and you must create a separate Haskell file with extension .hs and load it into ghci. It turns out that this is not entirely accurate.

(Refer Slide Time: 00:24)



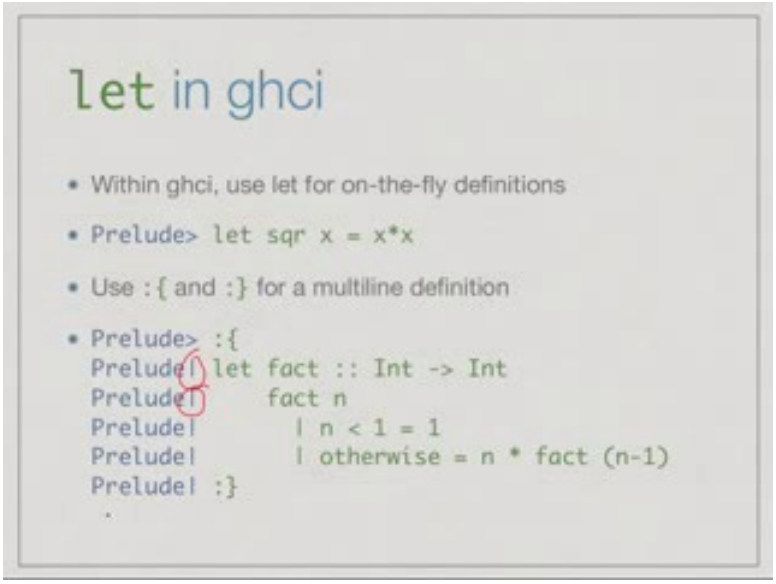
So, in order to use functions, definitions in ghci we need to learn a little bit more about Haskell. So, we have seen the use of the word 'where' in order to specify local definitions and functions. So, we can define a function in terms of some local definition and use these definitions in the function. An equivalent way of doing this is to use an expression called 'let'. So, we can instead of using where, we can say let diffx = x2 - x1, diffy = y2 - y1 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)



So, let _ in _ is a Haskell expression, is like if _ then _ else _ which is also Haskell expression. It can be used wherever an expression is allowed. At a level at which we are using Haskell, there is no significant difference between let and where, which is why we have 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 https://wiki.haskell.org/Let_vs._Where to find out some ways in which let differs from where.

(Refer Slide Time: 01:49)



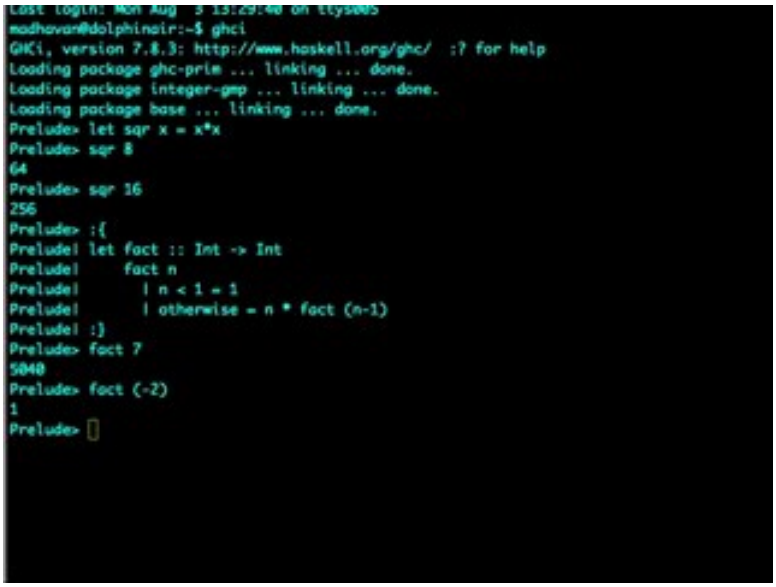
At this moment what is relevant for us is, that we can use let inside ghci to define functions on the fly. At a basic level we can use let to define single line values, so we can write something like let sqr x = x * x and now sqr will be available within ghci. So, therefore, it is definitely possible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

Of course, this is a single line definition. What if we want a multiple line definition, like say factorial which is defined separately for 0 and n. So, we can use this notation :{ and :} to enclose a multiple line definition. So, if we say :{, notice that you will find a slight change in the prompt that ghci gives you. Instead of the greater than sign >, it will give you something else perhaps a pipe symbol like this | .

Until you put a close brace and then the next line will return back to the usual form and in between, you can write a multiple line let for example, we can say something like,

let fact:: Int -> Int be the function where fact n, if n < 1 is 1, otherwise, it is n* fact (n-1).

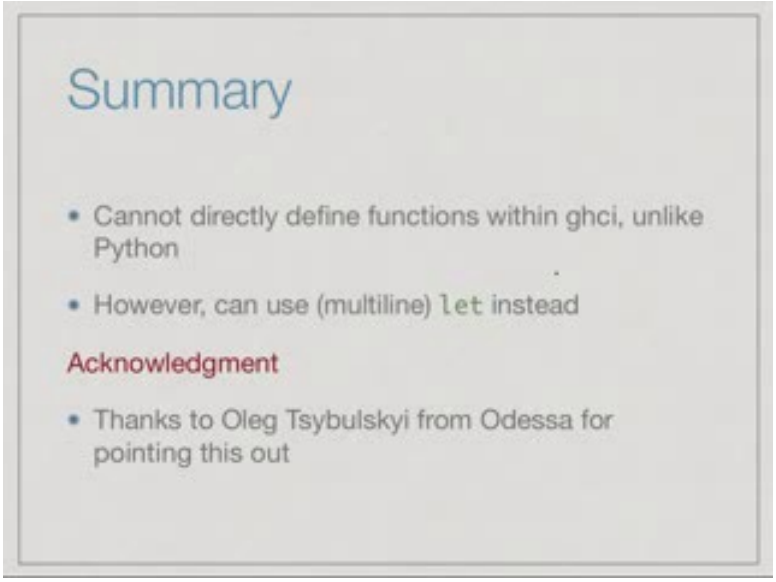
(Refer Slide Time: 03:08)



So, here let us actually do this to verify that this works. So, we say ghci, then we can say let sqr x = x*x . Now if we say sqr 8 for instance, we get 64. We say sqr 16 we get 256 and so on. Now, we want a multiline definition, we can prompt changes. Now, we can say let fact:: Int -> Int be the function given by fact n, such that n < 1 is equal to 1, otherwise, it is equal to n* fact (n-1) and then, we close this multi line definition, we get back to the old prompt.

Now, we say, what is fact 7, we get 5040, we say what is fact (-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)



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