Function Programming in Haskell

Prof. Madhavan Mukund and S. P. Suresh

Chennai Mathematical Institute

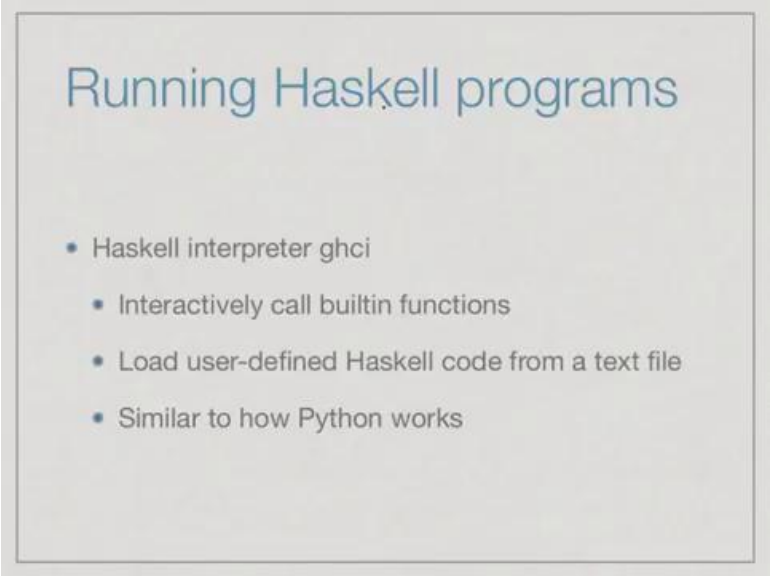
Module # 01

Lecture - 04

Running Haskell Programs

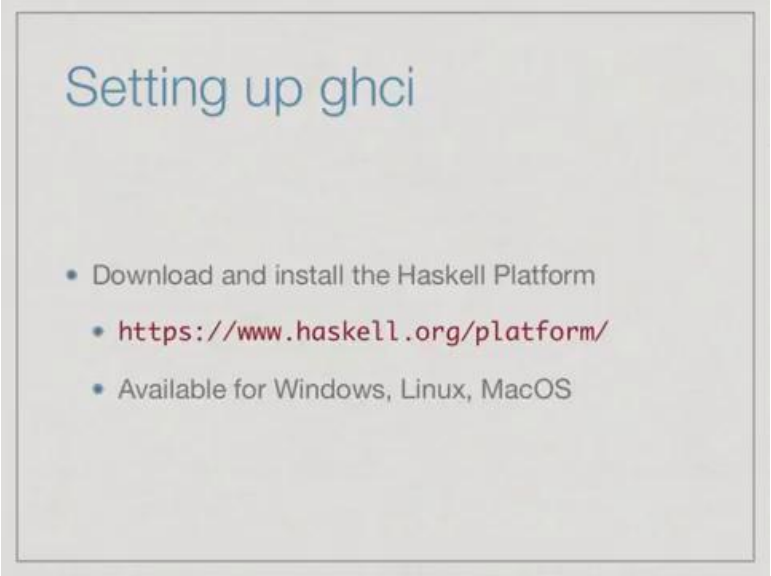
Having seen how to write Haskell programs, let us now look at how to run them on the computer.

(Refer Slide Time: 00:08)



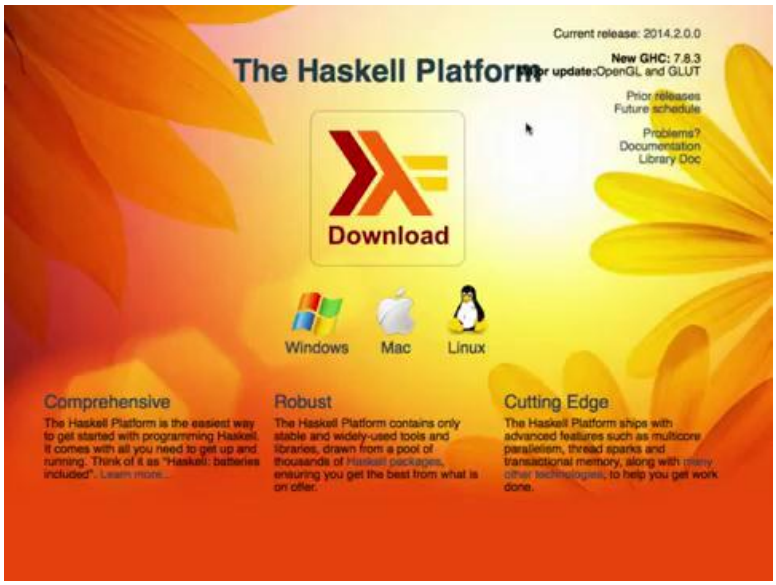
The easiest way to run a Haskell program is to use the Haskell interpreter ghci, using ghci will be very familiar to people who are use languages like python. So, ghci is an interpreter, you load ghci and then you can interact with it, you can directly called built in functions or you can load user defined Haskell code which you have previously entered into a text file.

(Refer Slide Time: 00:35)



Ghci is really available on the internet; the easiest way to get ghci working on your system is to download what is called the Haskell platform from the web URL given here.

(Refer Slide Time: 00:51)

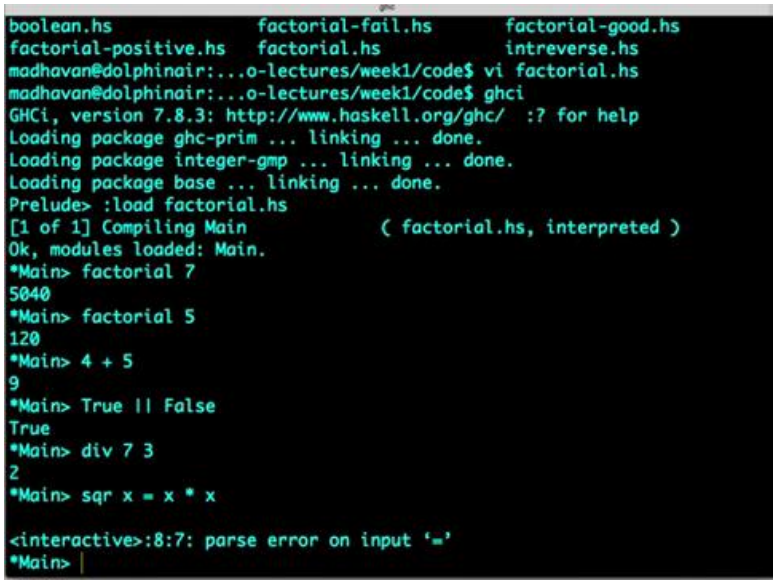


So, here is a screenshot of what the Haskell platform website looks like, as you can see you can download versions of the Haskell platform for any operating system for windows, for Mac OS and for Linux.

(Refer Slide Time: 01:10)

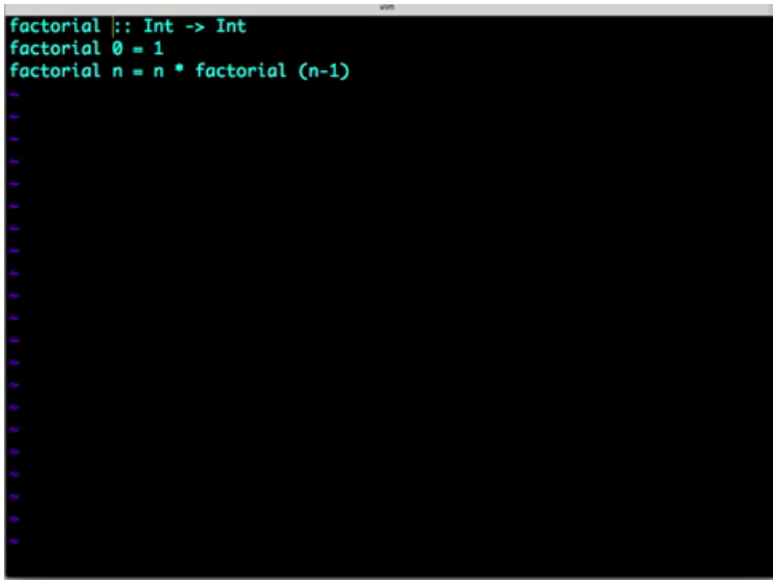
So, how do we actually use ghci? We first create a text file with the extension dot h s to indicate that it is the Haskell file containing your Haskell function definitions exactly as we wrote them in our pervious lecture, then we run ghci from the command line. Inside ghci you can then feed it further commands and the most basic command you need to know is a command to load Haskell code, this command is written as colon load followed by the file name, note the use of colon. So, colon is an indicator to ghci that it has to perform some internal action, anything without a colon is interpreted as a Haskell function that it must evaluate. So, in this way you can interactively call function from within ghci, let us look at some examples.

(Refer Slide Time: 02:01)



So, here is the command window in a Unix like system. So, in this we have already created some text files, but let us start from crashes, so supposing we want to create a version of factorial program. So, we open editor like vi or Linux. So, let us say vi factorial dot h s.

(Refer Slide Time: 02:22)

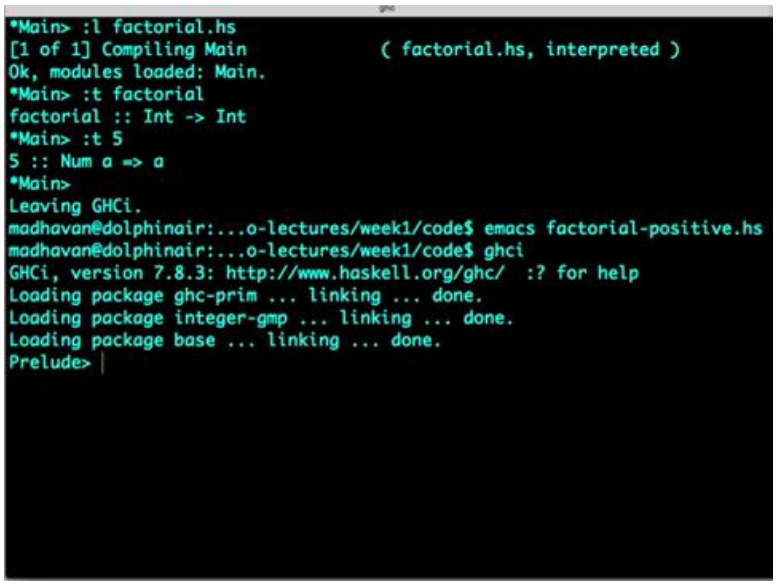


So, this is the first factorial program that we wrote, so there is an error, so let us fix that. So, we says that the type of factorial is int to int, factorial of 0 is 1 and factorial of n is n times factorial n minus 1. So, now, we can exit from the ((Refer Time: 02:37)), save the

file and load ghci. So, ghci will load up and give us a command prompt of it is own, it is in this case is with the prefix preview. Now, we can load the file that we have just created using the colon load command that we have discussed and now if I all is well you will get message like this saying it is okay. Having done this, we can now ask it to evaluate versions of factorial like factorial of 7 is 5040, factorial of 5 if you want to check a familiar numbers is 120 and so on.

You can also evaluate built in functions, for instance you can ask 4 plus 5. So, if you use it as a calculator, this is pretty much how you use it is things like python, you can say true or false or you can say div 7 3. So, it is very similar to the python interpreter if you use that, in that you can invoke any built in function or any function which you have defined in the file that has been loaded. As we will see one of the things that you cannot do in this, which you can do in a python interpreter should define functions here. So, I cannot write a new function here which says something like square x equal to x star x. So, this kind of a definition which is allowed in python, is not allowed in the interpreter, you have to write this in a separate file and then load.

(Refer Slide Time: 03:56)



So, as a short cut for instance you can use colon l instead of load, first of all you say colon h you get a list of all the possible commands, that you can do in ghci in case you want to investigate, but colon l is a short form. So, I can say colon l factorial dot h s and now it will just replace the old factorial by a new factorial. The other thing is that you

can query types, for instance I can now ask what is the type of factorial and it tells me that the type of factorial is int to int.

So, if you have a function which you not to type and sure of you can ask is, but you were that the types reported here can be more complicated in the types we have seen. For instance, you might ask what is the type of number 5, the number 5 you would expect as type int, but actually gives you this complicated thing which says num a and the different type of arrow with a, so we will see later what these things mean.

But, for function that you define you can verify that the types are indeed what is say they are. So, let us get out of the interpreter from moment and now let us look at this factorial that we wrote. So, I have got the same code in a file called factorial positive and this is to indicate that this factorial and this only the positive case. So, this is the same factorial we just use expect I have added now command a command in Haskell's start with two hyphens on the first tool columns of line. So, the first line is the command which says this is the factorial that does not handle make it inputs, which we already know, but let see what it does.

(Refer Slide Time: 05:39)

```
Prelude> :l factorial-positive.hs
[1 of 1] Compiling Main             ( factorial-positive.hs, interpreted )
Ok, modules loaded: Main.
*Main> factorial (-1)
^C
^D
^C
^A^Z
[1]+  Stopped                 ghci
madhavan@dolphinair:...o-lectures/week1/code$
madhavan@dolphinair:...o-lectures/week1/code$ kill %

[1]+  Stopped                 ghci
madhavan@dolphinair:...o-lectures/week1/code$
madhavan@dolphinair:...o-lectures/week1/code$ vi factorial-good.hs
madhavan@dolphinair:...o-lectures/week1/code$ 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> :l factorial-good.hs
[1 of 1] Compiling Main             ( factorial-good.hs, interpreted )
Ok, modules loaded: Main.
*Main>
```

So, if I go back to ghci and I load this factorial positive and I may ask it to do factorial say minus 1, then as we would expect it going to an infixed computation of factorial minus 2 minus 3 and so on. Because, we have not handle the negative phase and the only

way to get out of this is to physically break the program from run it, one way to do this is use control C or control D or nothing towards unfortunately.

So, we have to figure out a way to abort the program. So, now, instead we have written a better version with we have done already in the class, which said if n is less than 0 then you negate the factorial. So, now, so this is not a better ((Refer Time: 06:29)) this is a factorial with negative inputs.

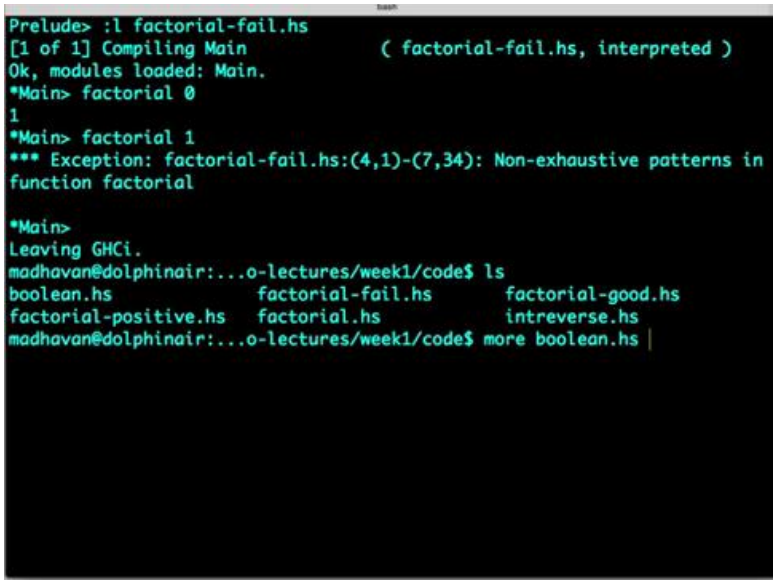
(Refer Slide Time: 06:45)

```
*Main> factorial (-4)
24
*Main>
Leaving GHCi.
madhavan@dolphinair:...o-lectures/week1/code$ ls
boolean.hs              factorial-fail.hs       factorial-good.hs
factorial-positive.hs   factorial.hs            intreverse.hs
madhavan@dolphinair:...o-lectures/week1/code$ more factorial-fail.hs
-- Factorial with pattern match failure

factorial :: Int -> Int
factorial 0 = 1
factorial n
  | n < 0  = factorial (-n)
  | n > 1  = n * (factorial (n-1))
madhavan@dolphinair:...o-lectures/week1/code$ 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>
```

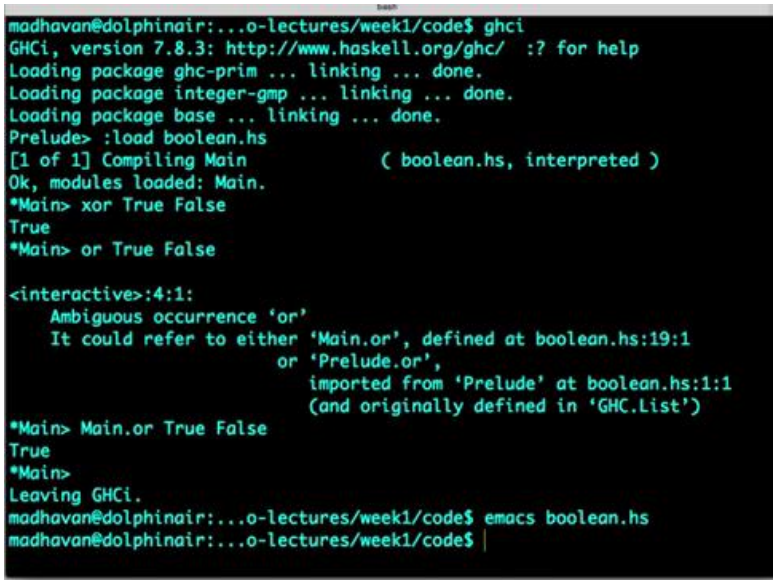
So, now, if I load this version and now they say factorial of minus 4 for instance then it converted to plus 4 and give me factorial of 4. Now, we also saw that we could have this kind of factorial function, where we have made a mistake. So, instead of n greater than 0 we have return n greater than 1 and this says that they will need not to be any case for factorial 1 words. So, let us see what happens when we load this.

(Refer Slide Time: 07:10)



So, now, if I say factorial of 0 it does not go in to the problematic case, so it is fine. But, if I say factorial of 1 then I get some kind of pattern match failure. So, it says an exceptions factorial fail and so on. So, the actual error message depends on the version of ghci, the error message return on the slides is slightly less expression in this. So, you gets some kind of error message and now finally, we have also return a version of to XOR and r that we did. So, we are XOR and an OR.

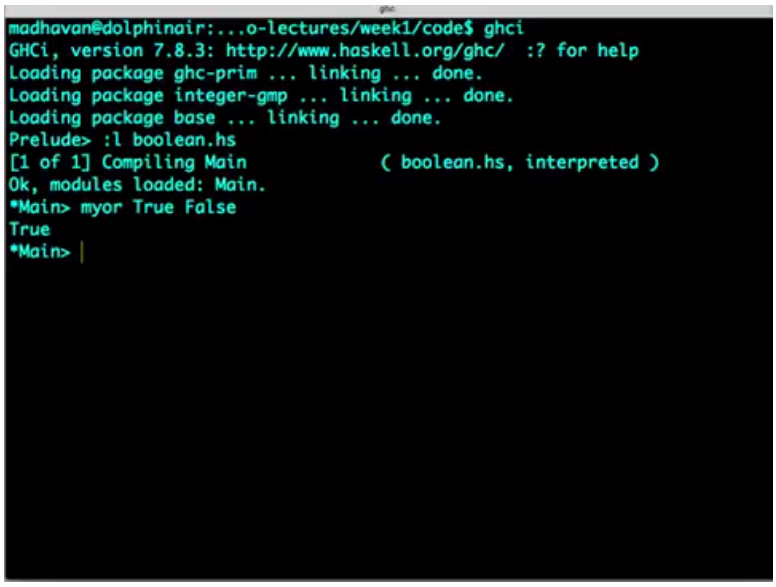
(Refer Slide Time: 07:54)



Now, here we have another problem which comes up and which you should be aware off. So, if I say load Boolean now if I say XOR a true false there is no problem and if I say OR of true false then I get an error message saying that is confuse by the world r, because the build in simple for OR is a two vertical lines, but OR as an English word can also be used in a Haskell program.

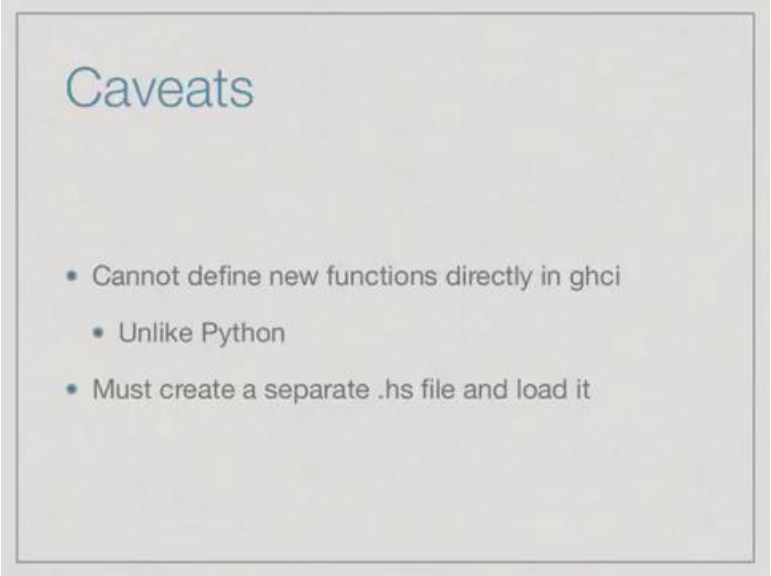
So, OR is a built in function and it says that the build in function OR is in conflict with the OR which is defined in this file. Now of course, one could say main dot OR and this main is the version that we have just loaded or even better than this is to actually go back and edit the file and use a new name. So, it is conventional to use the prefix my, so whenever you have a function whose name looks like a familiar function it is best to rename it which something which is disguise from the familiar function.

(Refer Slide Time: 09:03)



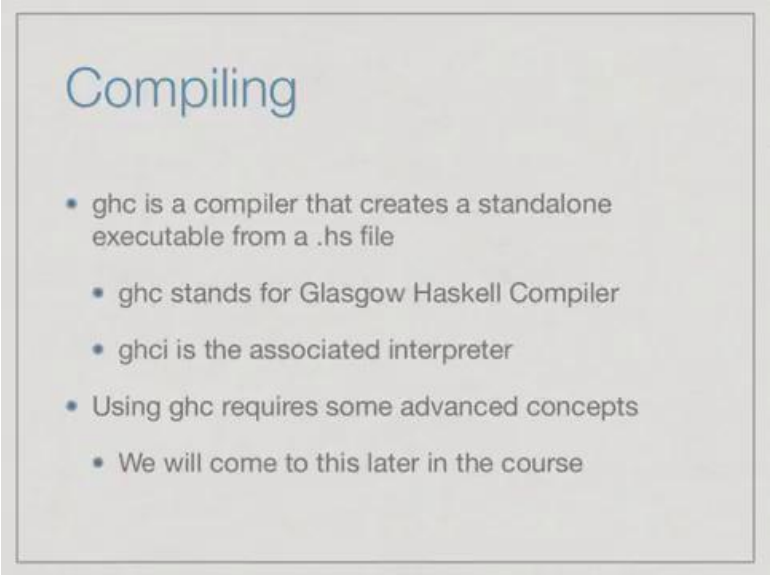
So, that now if I load this and say my OR true false it will work file. So, we have seen how to load files into that interpreter some kinds of error messages that you see you can also query types. So, it is very easy to use the interpreter and it is quite Haskell program tend to be small, so it is not very difficult to bigger then as we will see. So, please get familiar do download ghci on to your system and start playing around to it.

(Refer Slide Time: 09:37)



So, as we have mentioned when we were showing the use of ghci one of the differences between ghci and python is that you cannot create function definitions on the fly. So, within the interpreter you cannot define the new function, you must loaded to the separate function from an outside file. So, you must first created dot h s file and then loaded to be interpreter.

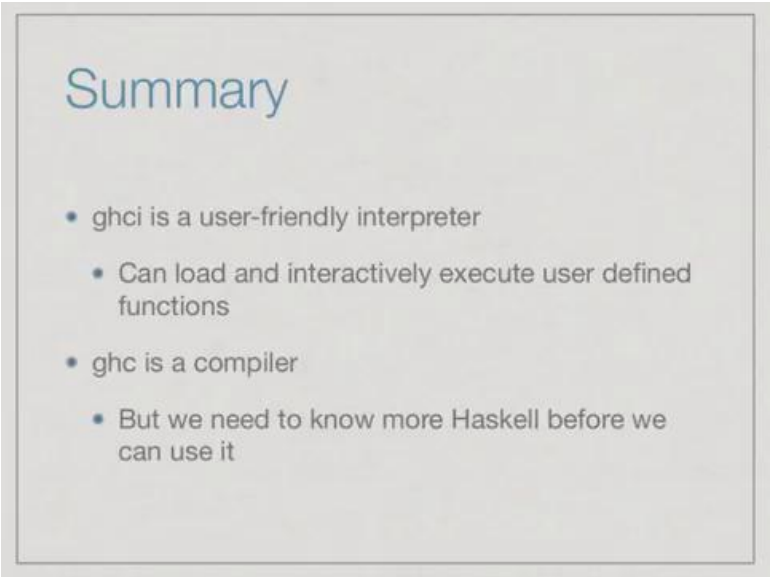
(Refer Slide Time: 10:00)



So, this is an interpreter, but the Haskell also conflict to fully fledged complier. So, ghc is a complier that creates and the executable file from a dot h s file. So, ghc stands for the

Glasgow Haskell complier and ghci is the associated interpreter which is the one that we have seen. Now, unfortunately we cannot use ghci directly on the Haskell function that we have been writing we needs some more advance complex which we will see as we go along. So, at some point we will start compiling or good, but for the moment we will just strict to ghci the interpreter.

(Refer Slide Time: 10:33)



So, to summaries the easiest way to run Haskell code is to use the interpreter ghci, we just quit user friendly and in which we can load and interactively execute user define functions, there is an associated complier for ghci, but we need to know more Haskell before we can use it.