**Function Programming in Haskell**

**Prof. Madhavan Mukund and S. P. Suresh**

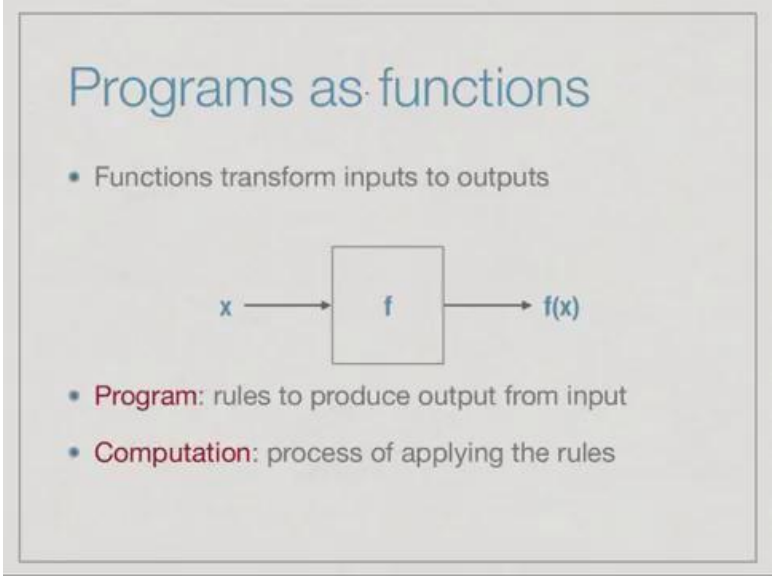
**Chennai Mathematical Institute**

**Module # 01**

**Lecture – 02**

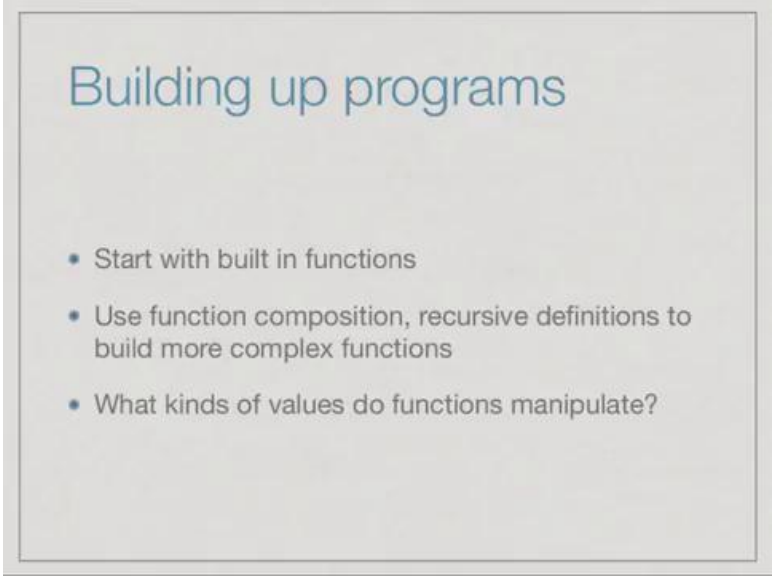
**Types**

(Refer Slide Time: 00:03)



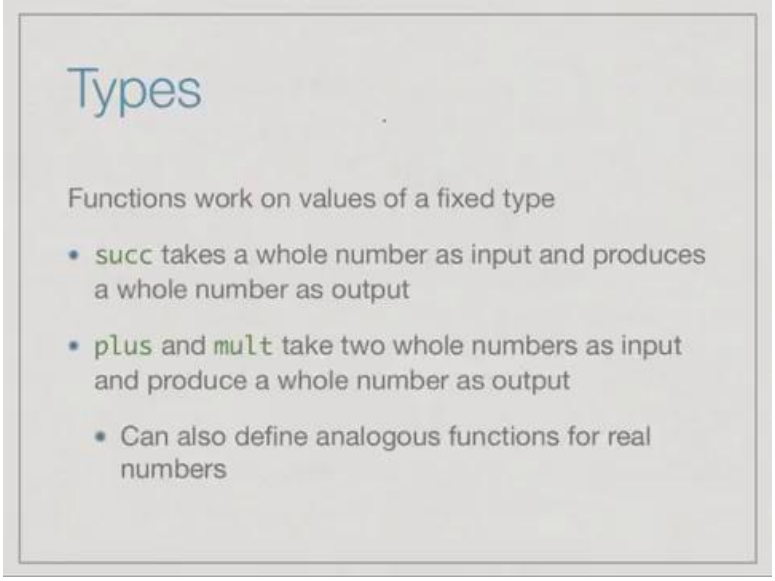

In the previous lecture, we saw that we would like to look at programs as functions that transform inputs to outputs. So, our program is basically a set of rules that describes how to produce a given output from an input. And a computation consists of applying these rules and transforming the input to the output were repeatedly rewriting expression based on the rules given.

(Refer Slide Time: 00:29)




So, the way we write a program is to start with some built in functions and values. And then, we use function composition and things like recursive definitions to built more complex functions from the basic built in functions and values given to us. So, another thing that we need to be aware of when we are dealing with functions is to look at the values that they manipulate, so these are what we called data types or just types.

(Refer Slide Time: 00:59)

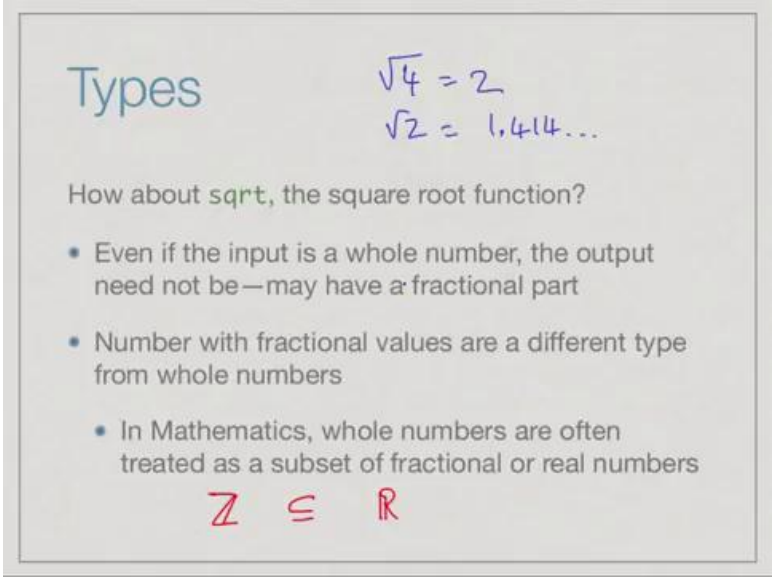



So, functions are defined over fixed input and output types. For instance, we saw last time the function successor which adds one, of the successful function that we define

assumes that inputs of the whole numbers 0 1 2 3 and as output it produces another whole number. We also looked at definitions of plus and multiply, so plus and multiply each took two whole numbers as input and produce the whole number as output.

Now, notice that if you add two whole numbers or you multiply two whole numbers you will necessarily get a whole number as an output. Now, you could also define similar functions plus and multiply for values that are not whole numbers or even successor. So, plus 1 could be defined for any value, but successor mean something successor means the next number. So, it make sense to say that the next whole number after 1 is 2, but it does not make sense to say that the next fractional number after 1.5 is 2.5. There is no next fraction number, which is why successor usually defined only for whole numbers. But, if you think of successor not a successor as plus 1, then we can define plus and multiply the general addition even for fractional numbers, but these will be different functions.

(Refer Slide Time: 02:16)

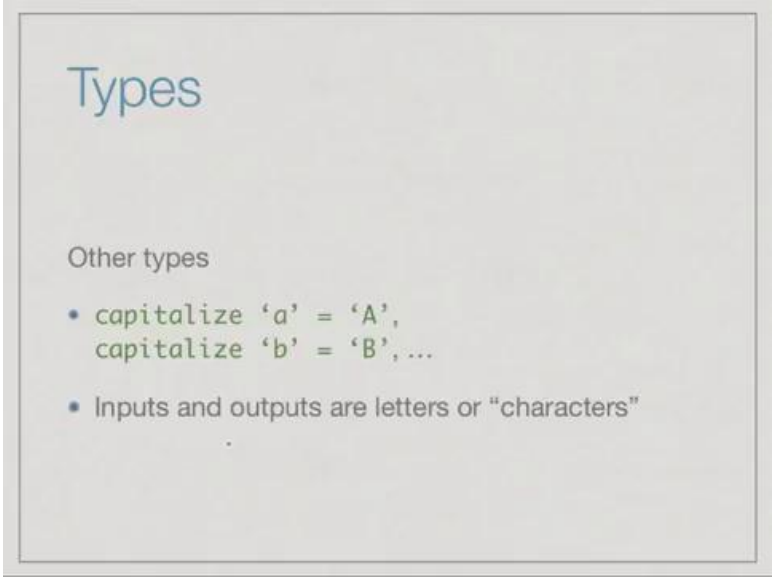



So, what about a function like square root that takes the square root of a number. So, here notice that even of the input is the whole number, the output need not be. The square root of 4 is 2, but the square root of 2 is in fact an irrational number, but at the very least it is the fractional number which is neither 1 nor 2. So, we have two different types of numbers that we deal with, we deal with this whole numbers or integers and we deal with numbers, which have fractional components.

The problem with fractional components is that this value after the decimal point can be arbitrarily long, could be infinite in general. And, so that to precisely describe a fractional value describe requires an infinite amount of information. So, now in mathematics typically you write sets like R for the real numbers and Z for the integers and you assume implicitly that the integers are those real numbers, which have a 0 fractional part.

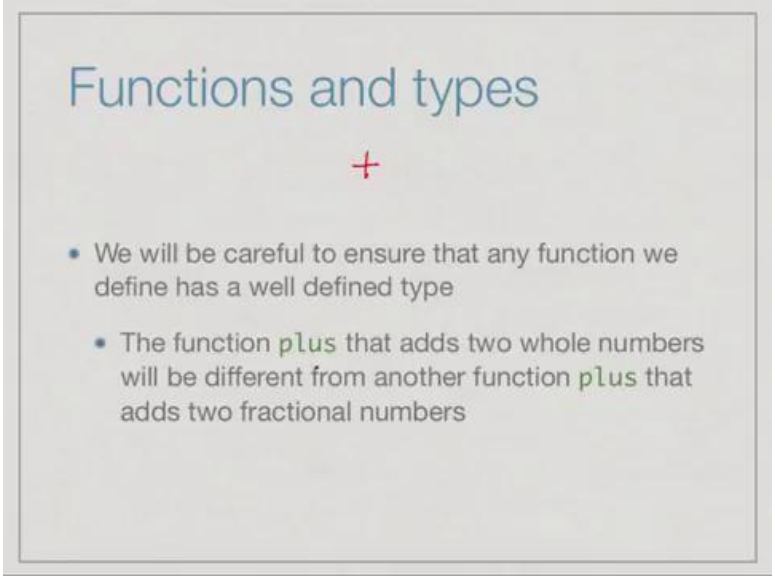
Now, when we write programs it turns out that representing integers and representing fractional numbers are very different and therefore, these are distinct types also. So, we will always differentiate between whole numbers and fractional numbers when we write functional programs, even though mathematically 1 is often seen as a subset with the other.

(Refer Slide Time: 03:40)




Now, many interesting programs manipulate values, which are not numbers. For instance, when you use a text editor or you enter a url in a browser, you are typing some text and then the program that is the web browser or your text editor has to interpret this text. So, characters are a very important type in programming. So, think of very simple function that we can write, something that takes an input letter and provided it is a valid letter of the alphabet not a punctuation mark or a number, it capitalize it. So, capitalize of small a is capital A, capitalize of small b is capital B and so on, so for such a function the input and the output consists of characters.

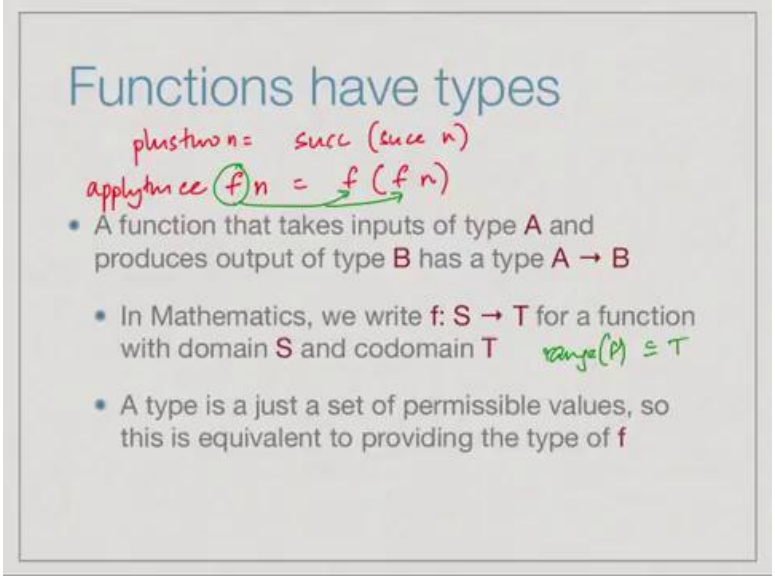
(Refer Slide Time: 04:22)




So, whenever we write functions in a function programming language you will have to ensure that the functions type is well defined. In fact, it will turn out that the functional language we are do not to use, which as Haskell will not allow us to write a function whose type is not defined. Now, not define can be many things, but one of the things it means is that the output must be uniform. So, you cannot have a function, which on some values produces an integer as now two and on some other values produces characterize.

So, this is to be kind of limitation that we will see, so all functions will have a well defined type. And therefore, the same function which say mathematically does a same thing like plus, which mathematically does a same thing for whole numbers and for fractional numbers will actually consist of two different functions they would maybe that for convenience we use a single like plus or the name p l u s to use to define both functions. But, impact in actually internally we are two different functions, because you have different types.

(Refer Slide Time: 05:23)




So, now, not only for the inputs and the outputs of functions have types the functions themselves have types. We will see that we can write functions, which take other functions as input and manipulate them for instance we can think of a function, so we saw that for instance plus 2 is successor of successor of a, so plus 2 of n success. Now, in general we could say that, why fix successor, why not say that we say apply twice.

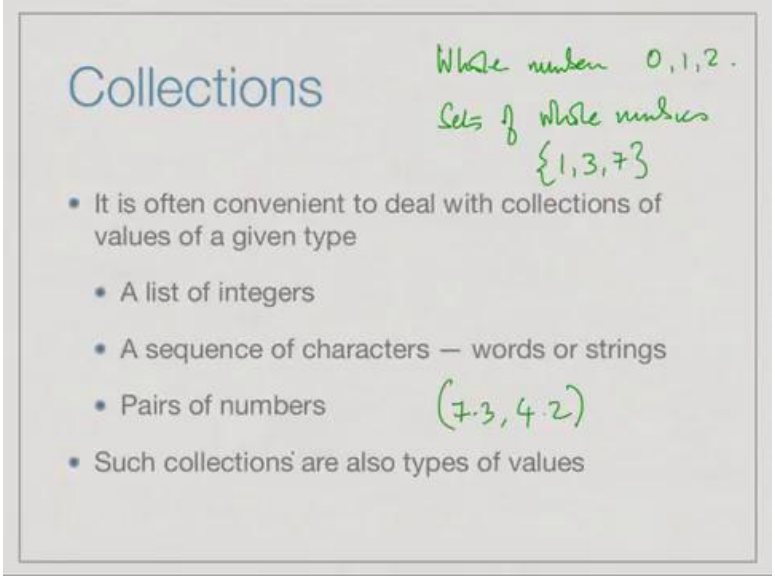
And we take as input a function and a number and we say the output if you are result to applying the function twice 2. So, here notice that the function itself is being supply to our new function and it is being use twice in the output. So, there is nothing that preventions from writing functions, which actually take other functions with arguments and now, let we assume that all functions must have well defined types for inputs and outputs functions must also have a types.

Now, the notion of the function type is not very mute wise in mathematics, if you write a function from a set S to a set T we write f colon S arrow T this means inputs come from S. So, S is often for the domain of the function f and outputs are from the set T. So, this is often called it co domain and in particular the range of the function is that subset of the co domain, which is actually reach by values from S. So, we have this notion of the input set and an output set.

So, a type of a function in a programming language is exactly the pictures take the input type and the output type and say transforms values of type A and values of type B. So, a

type is just to say of values, when we say we have the set of whole numbers, then it just means of the values from it as 0 1 2 3. We say we have set of characters when we have a set of symbols that we are allowed type will be have to fractional number, then a certain value that we can define decimal points. So, type is just a set of values and just like an mathematical notation we write a function could one set to another. We use the same notation denote the type of function and this is important, because one function can be actually fit to another function as input as use.

(Refer Slide Time: 07:48)

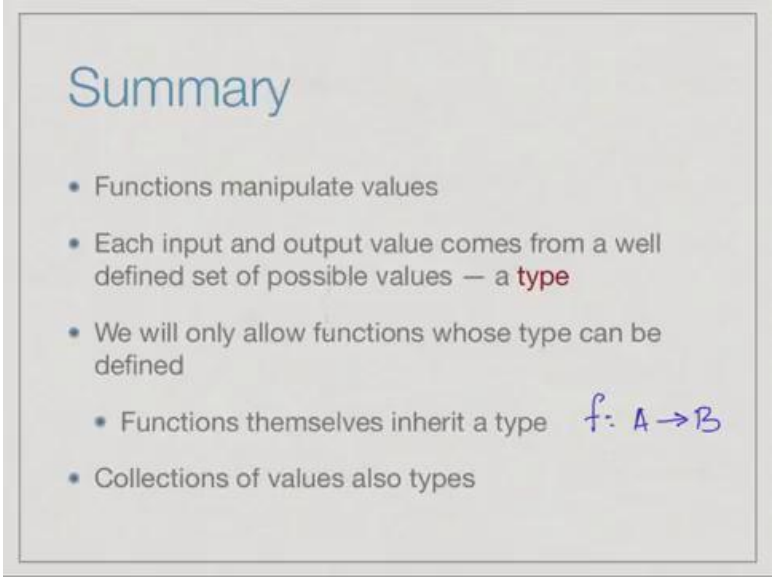



The other interesting motion is that of building collections of values. So, if you have whole numbers when we have individual value 0 1, now you can have sets of whole numbers for example mathematics. So, a set of whole numbers is a collection of whole numbers say 1 3 7 and we write it with some notation to indicates that is collections are may be a early brackets in commas.

So, similarly in programming we often want to deal with collections of values we welcome to talk about a sequence or a list of integers or when we type words, then a word consists of a number of characters of the sequence of characters are we might to take a word points in space, x y coordinates. So, an x y coordinate is just the pair of values like 7.3 and 4.2, so we want pairs of numbers. So, if you have types for the components, then we have a type for the whole, so we can say that I have a pair of fractional numbers I have a sequence of integers or I have a sequence of characters.

So, collections are also important types, so type remember, now just a set of omitted values. So, it tells you if I tell you the type then you can tell me whether a given value is legal for the type or not if I tell you that the type is whole numbers new as I ask you where seven point five is a whole number you can say is not a whole number. If, I ask you for a list of integer and ask you the list consist of not followed by 2 is a list of integers you say yes it is the list of integers and so on.

(Refer Slide Time: 09:26)




So, to summarize you are talking about functions that manipulate values, but the values must come from well specified sets or types. So, each input and output value comes from a well defined set of possible values and we will only allow functions definitions whose type is precisely specified. And given the types of the input and the output we saw that the function itself will had a type, which is from the input type to the output type.

And finally, you saw that it is important to have collections of values, so lists of integers or pairs of characters. So, we want to combine individual values are smaller values into larger values, which can store multiple values of the same type.