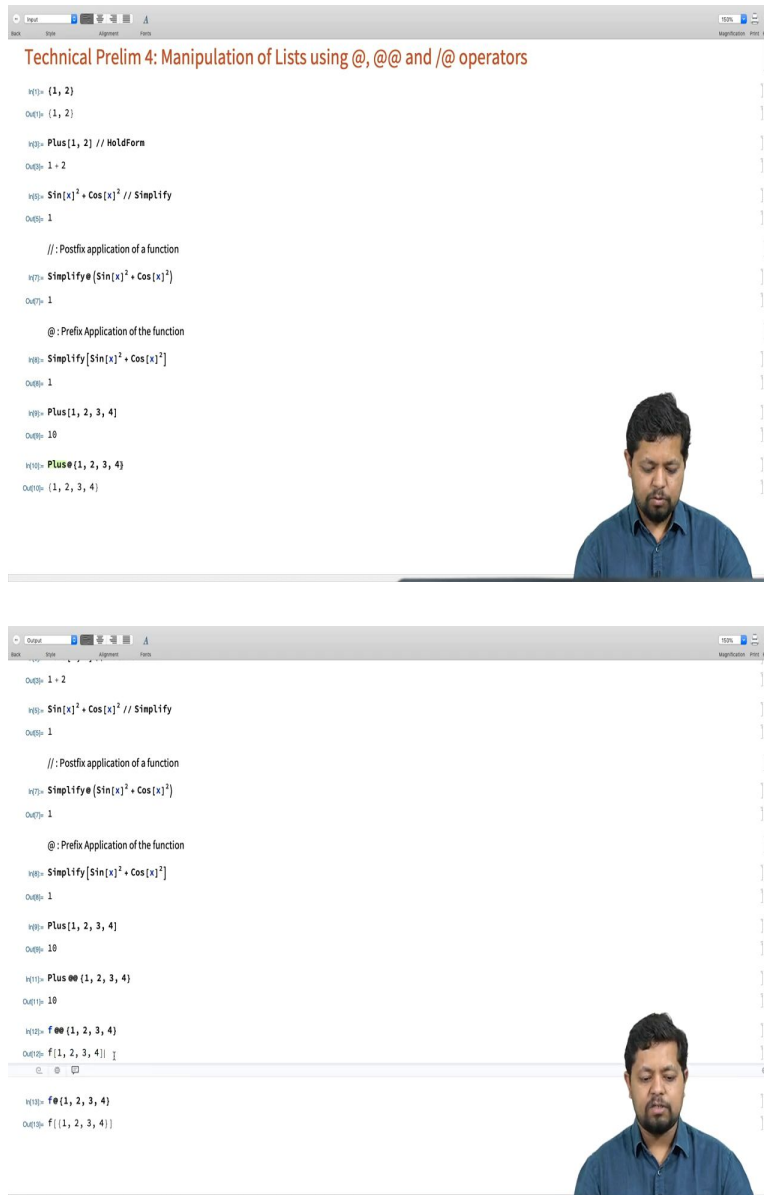


Physics through Computational Thinking
Dr. Auditya Sharma & Dr. Ambar Jain
Departments of Physics
Indian Institute of Science Education and Research, Bhopal
Lecture 24

Technical Prelim 4 – Manipulation of Lists using at, atat, -at operators

(Refer Slide Time: 0:27)



The image displays two screenshots of a Mathematica notebook interface. The top screenshot shows the input area with the following code:

```
Technical Prelim 4: Manipulation of Lists using @, @@ and /@ operators

M131: {1, 2}
O4E1X: {1, 2}

H5B1: Plus[1, 2] // HoldForm
O4E5D: 1 + 2
H5B1: Sin[x]^2 + Cos[x]^2 // Simplify
O4E5D: 1

//: Postfix application of a function
H7D1: Simplify@{Sin[x]^2 + Cos[x]^2}
O4E7X: 1

@: Prefix Application of the function
H8B1: Simplify[Sin[x]^2 + Cos[x]^2]
O4E8D: 1

H9B1: Plus[1, 2, 3, 4]
O4E9D: 10

H10B1: Plus@{1, 2, 3, 4}
O4E10D: {1, 2, 3, 4}
```

The bottom screenshot shows the output area with the following results:

```
O4E5D: 1 + 2

H5B1: Sin[x]^2 + Cos[x]^2 // Simplify
O4E5D: 1

//: Postfix application of a function
H7D1: Simplify@{Sin[x]^2 + Cos[x]^2}
O4E7X: 1

@: Prefix Application of the function
H8B1: Simplify[Sin[x]^2 + Cos[x]^2]
O4E8D: 1

H9B1: Plus[1, 2, 3, 4]
O4E9D: 10

H10B1: Plus@@{1, 2, 3, 4}
O4E10D: 10

H11B1: f@@{1, 2, 3, 4}
O4E12D: f[1, 2, 3, 4]

H12B1: f@{1, 2, 3, 4}
O4E13D: f[{1, 2, 3, 4}]
```

Welcome back to Physics through Computational Thinking. This is technical prelim 4 and we are going to discuss this time few technicalities of Holdform language of Mathematica and we

will learn about the implementation. What we are going to do today is we are going to learn about manipulation of lists using @, @@ and /@ operators. Let us say we have got a list of 1, 2, we want to apply some function on this list, let us say a Plus[1,2] simply means $1 + 2 = 3$.

If you want to see it, in it is, without evaluating, you want to see it in its natural form without evaluating it is $1 + 2$, or applying Holdform, on plus does not allow to evaluate and you can see the expression that is what Holdform does. The // operator is postfix for evaluating a function, so I want to evaluate any function let us say, I want to, let us say I want to do $\sin^2(x) + \cos^2(x)$ I want to simplify this.

So if I want, there are 2 ways of doing this, I can apply a postfix expression for simplify and that will simplify this to 1 so // is same as postfix operation. So, // is postfix application of a function. Similarly, if I want to apply a prefix application of function, for example, on the same expression, I will copy this expression over here. I can also apply simplify on this before and that apply simplify only on $\sin(x)$.

So, I should include, enclose this inside the square brackets. So now the simplify operation acts on entire bracket when I apply this, it gives me 1. This is equivalent to doing this up and closing $\sin^2(x) + \cos^2(x)$ inside simplify and evaluating it. So // is does postfix, at operator is prefix application of the function and finally, this is the standard form. Standard Form means the, the usually the way we apply functions that is we write the function and put its argument inside the square brackets.

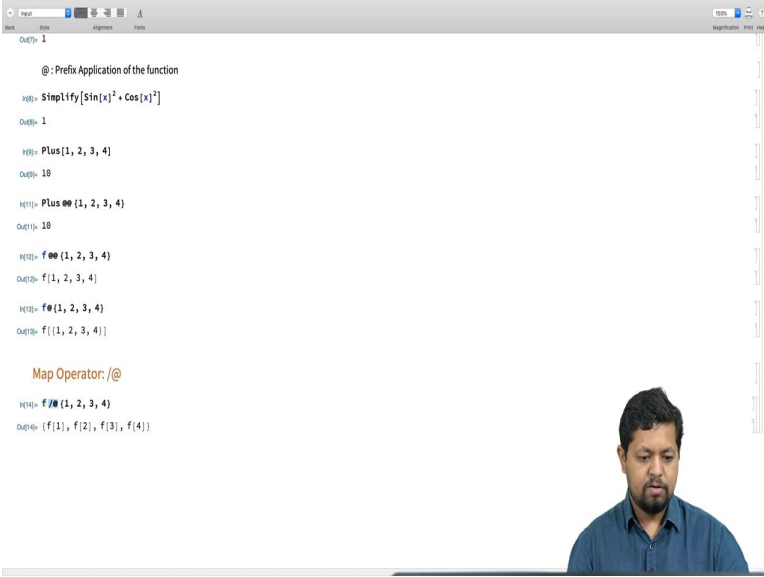
So, this is the standard form. Similarly, if I have got you know 2 arguments for example, as in the case of plus which is at 2 numbers or 3 numbers or 4 numbers a lot. For such a thing I can also do the following, I can define a list, which is elements 1, 2, 3, 4 and I can say apply plus to this. Know it is what happens when I do that. It does not change anything because it is applying plus the entire list.

What I want to do is I want to apply plus to the to the whole thing, and that gives me that gives me the evaluation of adding 1, 2, 3, 4. What I want here, the the @@ operator applies. Works

similar to at operator but takes each element of the list and takes that as an argument of the plus function. To see it more elaborately, let me take a function that I have not defined and let me apply that function f at this list 1, 2, 3, 4, it is equivalent to a function f applied on the arguments 1, 2, 3, 4.

So, at at operator works on lists and takes the all elements of list as argument of a function. Compare this with f@(1, 2, 3, 4). This will take the entire list 1, 2, 3, 4 as a single argument of the function f as shown over here. So, the function f has only 1 argument which is this list 1, 2, 3, 4. So, both at at and at prefix but at up takes, the whatever follows at whatever expression levels at is taken as the argument for f and for at at operator it must the whatever expression follows that list that follows that each argument is taken as, as the argument for the function.

(Refer Slide Time: 6:00)



```
IOE77: 1

@: Prefix Application of the function

HE8: Simplify[Sin[x]^2 + Cos[x]^2]
OUE8: 1

HE9: Plus[1, 2, 3, 4]
OUE9: 10

HE10: Plus@@[1, 2, 3, 4]
OUE10: 10

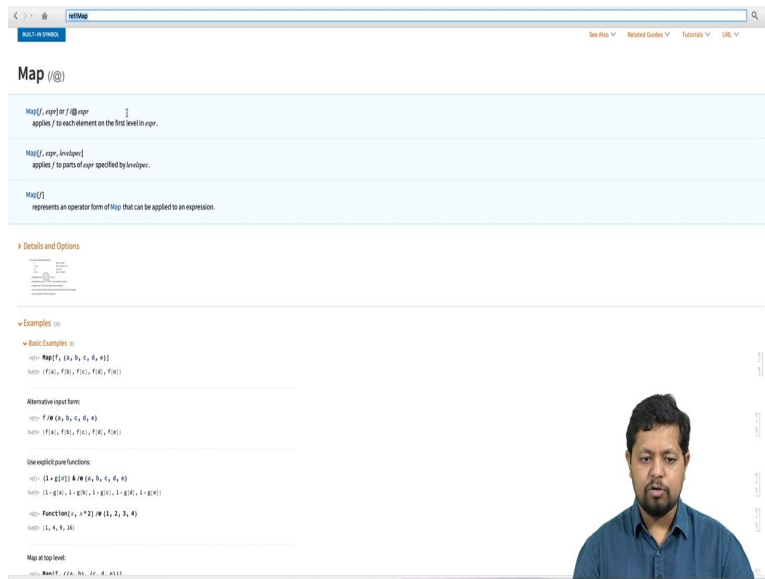
HE11: f@@[1, 2, 3, 4]
OUE11: f[1, 2, 3, 4]

HE12: f@[1, 2, 3, 4]
OUE12: f[[1, 2, 3, 4]]

Map Operator: /@

HE13: f/#[1, 2, 3, 4]
OUE13: {f[1], f[2], f[3], f[4]}
```

The screenshot shows a REPL window with a list of commands and their outputs. The commands include function applications like Simplify, Plus, and f, and the Map Operator /@. The outputs show the results of these operations, such as 1, 10, and a list of function applications.



Similarly, there is also a map operator. Map operator or the `/@` operator. If I take a list such as 1, 2, 3, 4, I can apply some function `f/@(1, 2, 3, 4)` the action that is generate me a list where the list contains f_1, f_2, f_3, f_4 , and so on. So, slash at operator is a map operator that is maps a list or array of elements to an array of function applied to each of the elements of the array. So, this map operator simply takes the function and distributes it over the list giving me another list where `f` is applied to all these different arguments.

If you want to read more about this, you go ahead, select this operator and press function `f_1` and they will give you the help for the map operator and you can read a lot about it. You can also use a map operator with the symbol `map`, `Map` rather than the `/@` but `/@` is much more convenient and will end up using several times. This was a quick overview of the map operator, `/`, `/@` operator, `@` operator and `@@` operator.