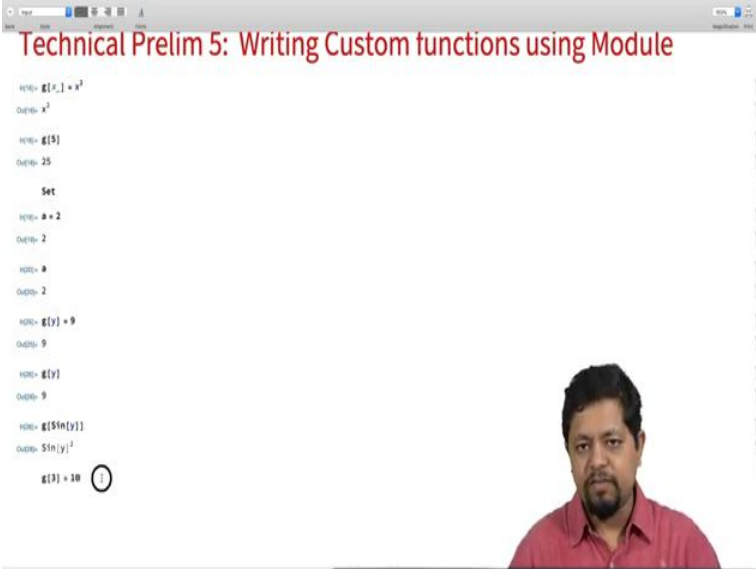


Physics through Computational Thinking
Professor Dr. Auditya Sharma
Dr. Ambar Jain
Department of Physics
Indian Institute of Science Education and Research, Bhopal
Lecture 26
Technical Prelim 5: Writing Custom Functions Using Module

Welcome, this is Technical Prelim 5, today we will learn how to write custom functions in mathematica, you have already seen a little bit of it and today we are going to learn how to write custom functions which involve multiple steps of calculation into a single function using the module construct. So, let us go ahead and get started with it.

(Refer Slide Time: 00:47)



```
Technical Prelim 5: Writing Custom functions using Module

In[1]: g[x_] := x^2
Out[1]: x^2

In[2]: g[5]
Out[2]: 25

Set

In[3]: a = 2
Out[3]: 2

In[4]: a
Out[4]: 2

In[5]: g[y] := 9
Out[5]: 9

In[6]: g[y]
Out[6]: 9

In[7]: g[sin[y]]
Out[7]: 5 sin[y]^2

g[3] = 10
```

Now, we are all familiar that if we want to define a function $g(x)$, then I should use the notation g in the square bracket I should write x and I should put underscore saying that x is a pattern and x is not just a symbol x , but the pattern x is what we are referring to. And at this point we will say equal to let us say x^2 , when we do that, I can call g by passing the value, so let me pass on 2 and you see that I will get $g[2] = 4$.

And you can try ahead and make $g[5]$ should be 25 and so on. Now, this is one way of defining the function and in this case, we can define a function in one go in one line, but sometimes it

requires function to have multiple steps, in that case this one line definition may not be enough and we will need to use module function. But before we go to module, let us also understand the subtle difference between set and set delayed.

Set is a built-in construct and if you go ahead and look at its documentation set is same as set can also be invoked by using equal to, for example, when I say $a = 2$ it is essentially I am calling the set command and saying set a as 2 that is $a = 2$, when I say $a = 2$, a is assigned a value 2 and when I call a in return I get the value 2.

The same thing happens for the function, the same thing happens for the function when I call the function, when I define the function $g[x]$ over here as this g is set as a function to be x^2 or whenever it sees a pattern which is what we as called as x over here, it is the pattern is squared and the value is returned.

Let us see the difference by defining as $g[y]$ and this time we will not give an underscore and we will say $g[y]$ is equal to let us say 7. And let us go ahead and do that and let us go and call $g[y]$ and you will see that will get 7. But at this time if I call $g[t]$ we will get t^2 , because $g[t]$ has not been defined with a specific value, $g[y]$ was defined with a specific value y , that means any time the symbol y is passed to g it will simply give me the value 7.


If I update $g[y]$ to let us say 9 and I call all $g[y]$ again execute this 9, I will get 9 back. I do the same thing for $g[2]$ or $g[2t]$ you see I will get $4t^2$ which is essentially the definition x^2 . So, let us understand this difference.

We have made explicit definition that whenever $g[y]$ is called it should be assigned the value 9. So, whenever it sees $g[y]$ we will get a value 9, but whenever it does not see $g[y]$, it sees anything else it will assume that as a pattern that is not been defined and go for a definition that have a definition of g for generic pattern. And then we will fall back onto this definition and for this definition x^2 has been defined and it will give x^2 .

And this is arbitrary, you can actually put in any expression over here, for example, I can put in $\sin[y]$ and it will give me $\sin^2[y]$. Similarly, you can go ahead and define $g[3] = 10$, you know

that $g[x]$ is x^2 , so $g[3]$ should be 9, but now I am over overloading that with $g[3]$ as a specific definition $g[3]$ being 10. So, let go ahead and define that.

(Refer Slide Time: 04:48)



```
Out[2]
In[ ]: g[y] = 9
Out[3]
In[ ]: g[y]
Out[4]
In[ ]: g[5 In[y]]
Out[5]
In[ ]: g[3] = 10
Out[6]
In[ ]: g[3]
Out[7]
In[ ]: g[3.0]
```

Now you see whenever I called $g[3]$, I will get a value 10, while if I call $g[2]$, I will get 4 and if I call $g[4]$, I will get 16, if I call $g[3]$, I will get 10, but if I call $g[3.0]$ I will get 9.0, because $g[3]$ was a specific pattern for integer 3, $g[3]$ was defined as 9 not as 10 not in general.

So, if I, so this is also known as function overloading for a given function you can define multiple definitions and depending on the context system will automatically understand which definition is being required and it will produce that definition, I will use that definition to produce your results.

Now, that we have understood that let us also understand difference between `set` and `SetDelayed`, if you go, if you want you can go ahead and look at the documentation for `set`, let us go ahead and look at `SetDelayed`.

(Refer Slide Time: 05:53)

The top screenshot shows the Wolfram Language documentation for `SetDelayed`. The title is `SetDelayed` [S]. Below the title, there is a description: "Assigns rhs to be the delayed value of lhs. rhs is maintained in an unevaluated form. When lhs appears, it is replaced by rhs, evaluated afresh each time." There is a circled 'S' in the 'Details' section. Below that, there are sections for 'Examples', 'See Also', and 'Tutorials'. The bottom screenshot shows a Wolfram Notebook interface. It contains several lines of code and their outputs. The code includes `Hold`, `Print`, and `SetDelayed` statements. The outputs are `9`, `5 ln|y|^2`, `10`, `9`, and `x^2`. The `x^2` output is circled.

`SetDelayed` is again a construct that can be simply called by colon equal to construct or colon equal to symbol. So, let me go ahead and open this documentation, if you look at the documentation of `SetDelayed` it says that `SetDelayed` or colon equal to is defined as I can be used as this, left hand side colon equal to right hand side.


Which means that assign the right hand side to be delayed value of left inside, it also says that right hand side is maintained in an evaluated form, well left hand side appears it is replaced by

right hand side evaluated afresh each time. So, difference between SetDelayed and set is the following.

If I define a new function let me say let me say, $gg[x]$ and this time I will use colon equal to that is I am going to use SetDelayed and I will say colon equal to as x^3 . Now, the effect of this is same as set or equal to colon equal to over here seems to have the same function as equal to but there is a subtle difference, equal to makes the evaluation of the right hand side immediately, so in this case, when I did the definition using equal to it evaluated the right hand side as x^2 , so x^2 was taken as a pattern x^2 and that value was immediately assigned to g .

Instead it SetDelayed that does not really happen this left hand side or right hand side is preserved as it is and left hand side is only assigned a value when gg is called. So, this is this is not become apparent or visible to you immediately, but this becomes important when the function can only be evaluated if a integer value is available.

(Refer Slide Time: 07:57)



```

In[ ]:= g[y]
Out[ ]:= 9

In[ ]:= g[5In[y]]
Out[ ]:= 5In[y]^2

In[ ]:= g[3] + 10
Out[ ]:= 10

In[ ]:= g[3]
Out[ ]:= 10

In[ ]:= g[3.0]
Out[ ]:= 9.

SetDelayed
In[ ]:= gg[x_] := x^3
Out[ ]:= x^3

In[ ]:= NIntegrate[x^2, {x, 0, y}]
Out[ ]:= NIntegrate[x^2, {x, 0, y}]

```

The screenshot shows a Mathematica notebook interface. The code cell contains several lines of Mathematica code. The first part defines a function `g` using `g[y]` and `g[5In[y]]`, and then uses `g[3] + 10` and `g[3]` to show that the right-hand side is evaluated immediately. The second part uses `SetDelayed` to define `gg[x_] := x^3`, and then shows that `gg[3]` evaluates to `x^3`. The final part shows `NIntegrate[x^2, {x, 0, y}]` being evaluated, with a red error message: `NIntegrate::x-y is not a valid limit of integration.` A small circle is drawn around the error message.

```

In[ ]:= g[y]
Out[ ]:= 9

In[ ]:= g[Sin[y]]
Out[ ]:= 51n[y]^2

In[ ]:= g[3] + 10
Out[ ]:= 10

In[ ]:= g[3]
Out[ ]:= 10

In[ ]:= g[3.0]
Out[ ]:= 9.

SetDelayed
In[ ]:= g[x_] := x^3
Out[ ]:= x^3

In[ ]:= NIntegrate[x^2, {x, 0, y}]
Out[ ]:= NIntegrate[x^2, {x, 0, y}]

```



```

In[ ]:= g[Sin[y]]
Out[ ]:= 51n[y]^2

In[ ]:= g[3] + 10
Out[ ]:= 10

In[ ]:= g[3]
Out[ ]:= 10

In[ ]:= g[3.0]
Out[ ]:= 9.

SetDelayed
In[ ]:= g[x_] := x^3
Out[ ]:= x^3

In[ ]:= g[y_] = NIntegrate[x^2, {x, 0, y}]
Out[ ]:= NIntegrate[x^2, {x, 0, y}]

```



For example, let us go ahead and do numerical integration of x^2 and where x goes from 0 to y and you see that when I call this that is going to complain, because the y is not a numerical limit, it says $x = y$ is not a valid limit of integration, y is not a number, on the other hand if this was 1, this will evaluate to a number, but if I give it something that has not been defined as a number, it is going to complain that this is not work.

But if that is my need I can go ahead and define a function that we will say $g2$ and this time I am going to say $g2[y] = NIntegrate$. Now, let me go ahead and do that and you see that, it is going to

complain, because it is going to evaluate the right-hand side and it is unable to evaluate the right hand side, because y is not a number.

So, it is going to complain that it is not able to evaluate the right-hand side and it is not able to do this do this properly.

(Refer Slide Time: 09:25)

```
In[1]: g[y_] := Sin[y]
Out[1]: Sin[y]^2

In[2]: g[3] + 10
Out[2]: 10

In[3]: g[3]
Out[3]: 10

In[4]: g[3.0]
Out[4]: 9.

SetDelayed
In[5]: g[x_] := x^3
Out[5]: x^3

In[6]: g2[y_] := NIntegrate[x^2, {x, 0, y}]
Out[6]: NIntegrate[x^2, {x, 0, y}]
-- NIntegrate::x-y: x or y is not a valid limit of integration.
Out[6]: NIntegrate[x^2, {x, 0, y}]
```

```
Out[6]: 9

In[7]: g[y_] := Sin[y]
Out[7]: Sin[y]^2

In[8]: g[3] + 10
Out[8]: 10

In[9]: g[3]
Out[9]: 10

In[10]: g[3.0]
Out[10]: 9.

SetDelayed
In[11]: g[x_] := x^3
Out[11]: x^3

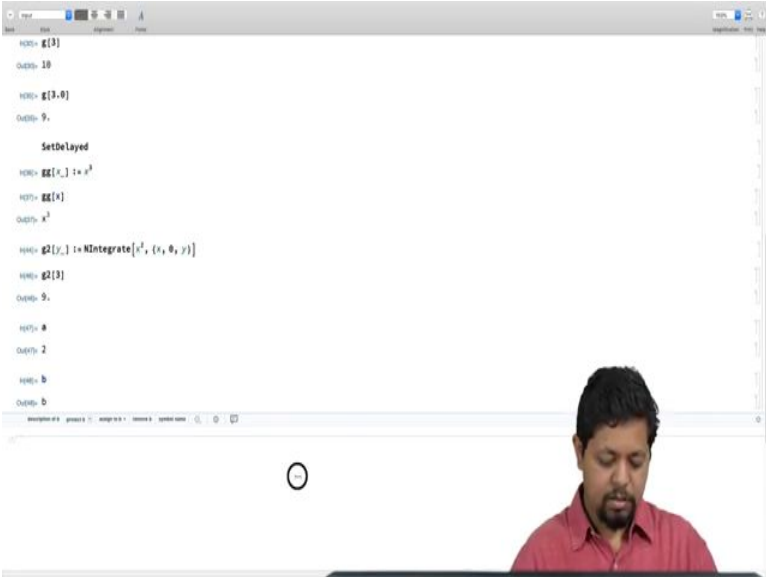
In[12]: g2[y_] := NIntegrate[x^2, {x, 0, y}]
Out[12]: NIntegrate[x^2, {x, 0, y}]
-- NIntegrate::x-y: x or y is not a valid limit of integration.
Out[12]: NIntegrate[x^2, {x, 0, y}]
```

So, now if I go ahead and call $g2[1]$, so now I can solve this problem by define a set delayed, so I can go ahead and set $g2[y] := NIntegral$ of this and now if I go ahead it says that it has no problem with that because it is not try to attempted to evaluate the right-hand side.

At this point I can now go ahead and call $g2[2]$ or $g2$ if I call $g2[y]$ for example, where y is just a symbol, it is going to complain as before because now it is time to attempt to evaluate the right hand side. But if I pass it on numerical value and integrate we will be happy and it is going to be fine with the value that is passed on to it and it is going to evaluate it to a number.

So, I hope you have understood the difference between equal to and colon equal to SetDelayed and certain SetDelayed. Let us go ahead and think about constructing more complex functions which cannot be done in a line, so for such things and also when we require internal variables or local variables that are not defined that we do not want to be defined the global context.

(Refer Slide Time: 10:55)



```

In[1]: g[3]
Out[1]: 18

In[2]: g[3.9]
Out[2]: 9.

SetDelayed
In[3]: gg[x_] := x^3
Out[3]: gg[x]
Out[3]: x^3

In[4]: g2[y_] := NIntegrate[x^2, {x, 0, y}]
In[5]: g2[3]
Out[5]: 9.

In[6]: a
Out[6]: 2

In[7]: b
Out[7]: b

```

By global context, we mean that the function g exist in the global context function gg that we have define exist in the global context and function $g2$ that we have defined exist in the global context and so on.

Similarly, we also defined a , which was assigned a value 2 it exists in a global context, while we have not yet different b and does not have a value in the global context. So, suppose we want to

define a function which involves multiple steps, then we need to use a construct like module, so I will show you how to use module.

(Refer Slide Time: 11:31)

The image displays two screenshots of a Mathematica notebook interface. The top screenshot shows the following code and outputs:

```
Out[10]:  
In[10]: g[3,0]  
Out[9]:  
SetDelayed  
In[10]: g[x_] := x^2  
Out[9]:  
In[10]: g[x]  
Out[9]: x^2  
In[10]: g[1,1] := NIntegrate[x^2, {x, 0, y}]  
Out[9]:  
In[10]: g[3]  
Out[9]:  
In[10]: a  
Out[9]: 2  
In[10]: b  
Out[9]: b  
In[10]: ff[x_] := Module[{x}, x^2]  
Out[10]: x^2  
In[10]: ff[3]  
Out[10]: x^2
```

The bottom screenshot shows the same code but with a different output for the last two lines:

```
Out[10]:  
In[10]: g[3,0]  
Out[9]:  
SetDelayed  
In[10]: g[x_] := x^2  
Out[9]:  
In[10]: g[x]  
Out[9]: x^2  
In[10]: g[1,1] := NIntegrate[x^2, {x, 0, y}]  
Out[9]:  
In[10]: g[3]  
Out[9]:  
In[10]: a  
Out[9]: 2  
In[10]: b  
Out[9]: b  
In[10]: ff[x_] := Module[{x}, x^2]  
Out[10]:  
In[10]: ff[3]  
Out[10]: 100
```

Module construct is takes 2 arguments, one is the local variables which are given as a list of inside a pair of curly brackets and the expression. So, in this case let me go and say x^2 and this point I will say x is a local variable. Let us, go ahead and evaluate that and you see it gives me some sort of gibberish.

Because in this context x is a local variable, module was given some number, this is that number, of that call of the module and as a consequence the variables that is internal to mathematica system is x dollar followed by this number which is being squared. So, this does not really give me anything sensible and it should not, because module is used to define functions.

So, what I need to do is, I need to define a new function, let us say $ff[x]$ and let me go ahead and say equal to x^2 and let me execute it and you see this time again, I will get some gibberish of this form which does not seem to be very interesting, but let me go ahead and anyway calculate $ff[3]$ and see what happens, you see that $ff[x]$ was assigned the value, this so it is so immediately the right hand side was evaluated and $ff[x]$ was assigned a value x times this squared.

And now when I pass an argument 3 to it, it does not really work, because the right hand side got evaluated. So, when I am using a module construct, I really want to use a colon equal to and I in this case I do not attempting to use same x as an argument or the function ff and the local variables so this complaining I can go ahead and remove this keep it empty go ahead and execute it. And now when I call $ff[3]$ it will evaluate it to 9. Let me try something else $ff[10]$ evaluate to 100 and so on.

Now, this is the basic usage of ff of module construct and that does not really illustrate much, let me go ahead and take an example at this point. Let us say I want to calculate the speed given distance and time.

(Refer Slide Time: 14:03)

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

```
SetDelayed
H436) gg[x_] := x^2
H437) gg[x]
O437) x^2
H440) g2[y_] := NIntegrate[x^2, {x, 0, y}]
H441) g2[3]
O441) 9.
H472) a
O472) 2
H482) b
O482) b
H532) ff[x_] := Module[{i, x^2}
H542) ff[10]
O542) 100
H562) speed[dist_, time_] := Module[{spd},
  i)   dist
      spd =  $\frac{\text{dist}}{\text{time}}$ ;
      Return[spd];
```

The bottom screenshot shows the same code with the function call and output:

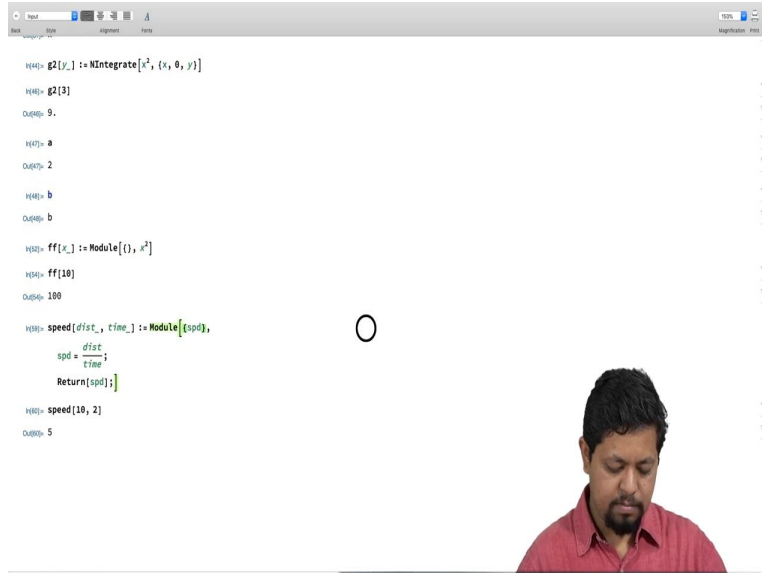
```
H562) speed[dist_, time_] := Module[{spd},
  dist
  spd =  $\frac{\text{dist}}{\text{time}}$ ;
  Return[spd];
O562) Return[ $\frac{\text{dist}}{\text{time}}$ ]
H572) speed[10, 2]
O572) 5
```

So, now my function speed is going to have 2 arguments that is distance and time and I will say colon equal to module and this time I will define an internal variable and I will call that internal variable as spd for short for speed and then for the body, I will write spd equal to distance divided by time and I will say return spd, returned is going to return the value of that of the speed that has been calculate.

Now, there multiple steps or multiple statements involved in the body, this is this, these 2 highlighted statements are the body of the module, there multiple statements involved in this

body and therefore they have to be separated by semicolon. So, let me go ahead and execute that, so at that exclusion defines speed because speed has been defined with the SetDelayed, notice that if I define it with without that it is going to give me this, which will become basically useless at least that is what I think, now it works. But that is not very important and useful, therefore most applications.

(Refer Slide Time: 15:41)



```

In[41]: g2[y_] := NIntegrate[x^2, {x, 0, y}]
Out[41]: 9.

In[42]: a
Out[42]: 2

In[43]: b
Out[43]: b

In[44]: ff[x_] := Module[{t, x^2}
Out[44]: ff[10]
Out[44]: 100

In[45]: speed[dist_., time_] := Module[{spd},
Out[45]: speed[10, 2]
Out[45]: 5

```

So, let me go ahead and make definitions colon equal to and so you have to understand the difference in speed is a local variable that is a spd is a local variable and distance and time are arguments of the function that I am defining speed. So, first I calculate spd equals distance over time and then I returned speed and now when calculate speed for distance equal to 10 units and time equal to 2 units, I get a value of 5.

Now, it may be so that you require to calculate this speed when the distance and you want the answer in meter per second while the distance is given in kilo meters and time is given in hours, so in that case you need to do the following, let us say distance is given in kilo meters and time is given in hours and I want to get the speed in meters per second.

(Refer Slide Time: 16:40)

The image shows two screenshots of a Mathematica notebook interface. The top screenshot shows the following code and output:

```
In[1]: g2[x_] := NIntegrate[x^2, {x, 0, y}]
In[2]: g2[3]
Out[2]: 9.
In[3]: a
Out[3]: 2
In[4]: b
Out[4]: b
In[5]: ff[x_] := Module[{t}, x^2]
In[6]: ff[10]
Out[6]: 100
In[7]: speed[dist_, time_] := Module[{timeInsec, distInmeters, spd},
  timeInsec = 3600 time;
  distInmeters = 1000 dist;
  spd = distInmeters / timeInsec;
  Return[spd];
]
In[8]: speed[10, 2]
Out[8]: 5
```

The bottom screenshot shows the same code with additional output for the speed function:

```
In[9]: speed[72, 2]
Out[9]: 10
In[10]: timeInsec
Out[10]: timeInsec
In[11]: distInmeters
Out[11]: distInmeters
```

Handwritten annotations include a circle around the 'time_' argument in the function definition in the top screenshot, and a circle around the 'time' argument in the function call in the bottom screenshot.

So, step by step, if I want to do this, I would probably want to do something like that, I like to get time in seconds and I want to get distance in meters and then I want to define speed, so first step would be to calculate time in seconds, given the time is in hours.

So I will say the time argument that has been given in hours rather I should say time in second is equal to time in hours times 3600, 3600 put a semicolon, then I need to calculate distance in meters and this time I want to multiply 1000 to the this argument plus semicolon then I want to calculate speed as distance in meters divided by time in seconds and the return speed.

So, now you see that if the speed is work with the familiar numbers if speed is 36 kilo meter, the distance 36 kilo meter and time is 1 hour you know the speed is 10 meter per second and this does work and you can check again if I make that 72 in 2 hours it is going to be 10 again and likewise you can go ahead and play with this example.

So, now you see that when I want to do a calculation in multiple steps, I can define custom functions using the module construct, I can use n number of local arguments and those local arguments are used inside over here local variables time in seconds, distance in meters and speed at local variable.

You can see that time in second is not defined in the global context, there is no value of time in second in the global context. Similarly, distance in meters has no value in the global context its internal variable only to this module and outside this module there is no value assigned to assigned to that.

So, that way when you are working with a complex calculation where multiple variables you need to define, but you do not want them to get a global context, you want them to be in a local context, you can put inside a module do multiple steps of calculation and finally return 1 or 2 values that you want to return and define all this entire module as a function with a SetDelayed operation.

All right, I hope you understood how to use module, will use in some examples that we will going we are going to use in the course.