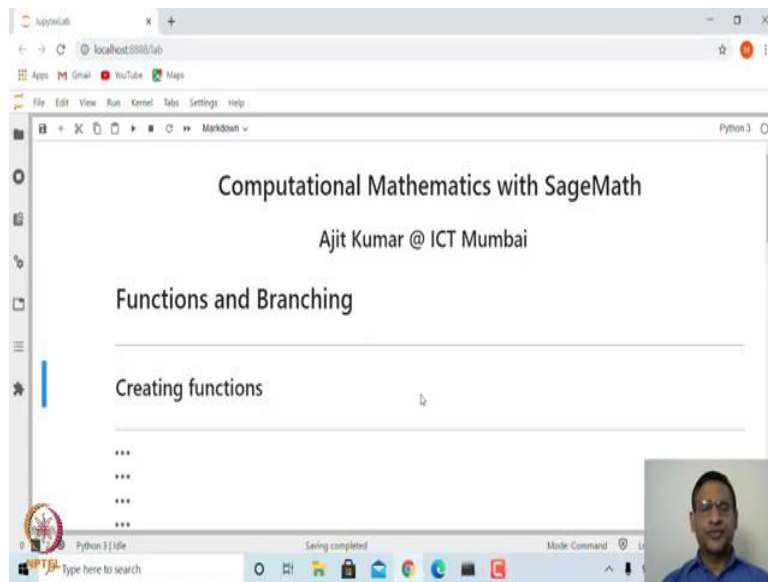


Computational Mathematics with Sagemath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Lecture – 06
Functions and Branching

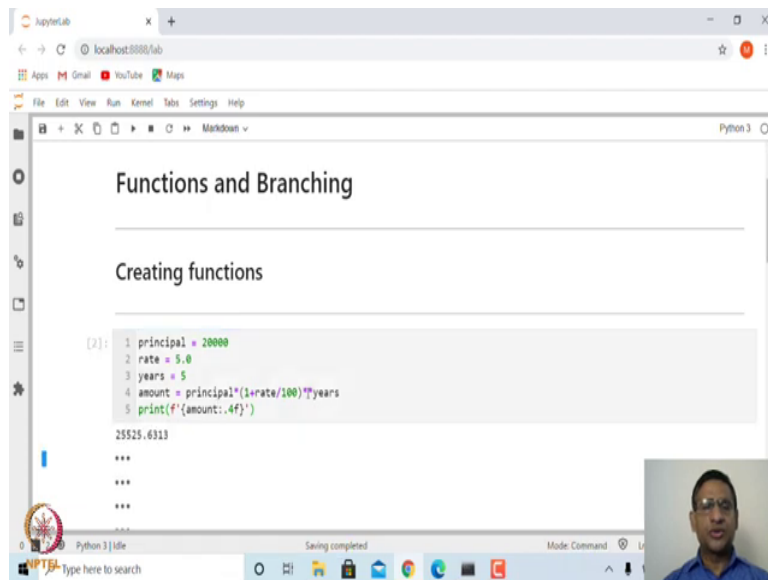
Welcome to the 5th lecture on Computational Mathematics with SageMath. In this lecture, we will look at how to create our own functions in Python. We should also look at how to make decisions based on certain conditions, what is called 'Branching' in Python.

(Refer Slide Time: 00:38)



So, let us get started. First let us look at creating user defined functions. Recall that we have done computation of Compound Interest, let us just look at what we did.

(Refer Slide Time: 00:58)

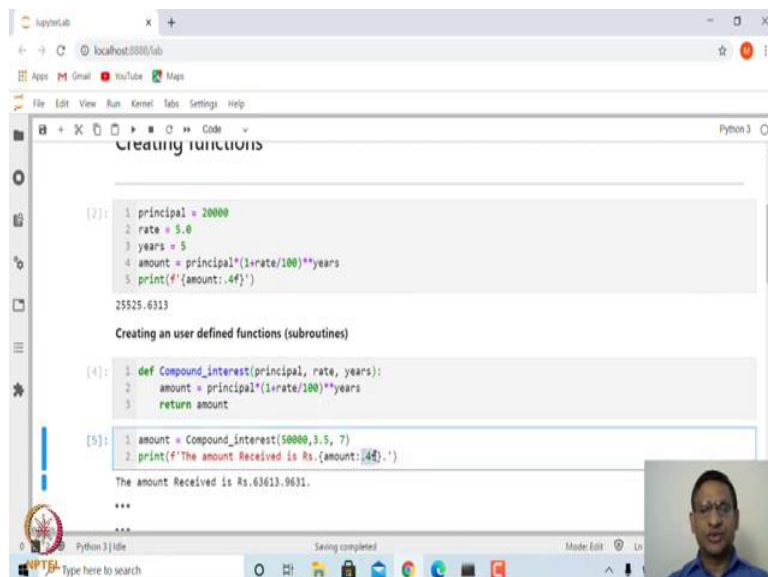


```
[2]: 1 principal = 20000
2 rate = 5.0
3 years = 5
4 amount = principal*(1+rate/100)**years
5 print(f'{amount:.4f}')

25525.6913
***
***
***
```

Suppose there is a principal amount of 20000 which is invested at the rate 5% for 5 years. Then, the amount after 5 years, we calculated using this formula, P into 1 plus r by 100 to the power t . Now, suppose we want to create a function for computing the amount returned, when it is invested under compound interest. How do we do that? It is fairly simple.

(Refer Slide Time: 01:39)



```
[2]: 1 principal = 20000
2 rate = 5.0
3 years = 5
4 amount = principal*(1+rate/100)**years
5 print(f'{amount:.4f}')

25525.6913

Creating an user defined functions (subroutines)

[4]: 1 def Compound_interest(principal, rate, years):
2     amount = principal*(1+rate/100)**years
3     return amount

[5]: 1 amount = Compound_interest(20000, 5, 5)
2 print(f'The amount Received is Rs.{amount:.4f}.')

The amount Received is Rs.25525.6913.
***
***
```

What we need to do is that, we need to simply write `def` and space then, write the name of the function. I am giving the name `compound underscore interest` and then, these are the arguments. In this case, the arguments we want is the investment, that is, the principal, the rate at which

the interest rate is calculated and the number of years for which the investment is made and then, calculate the amount using the same formula.

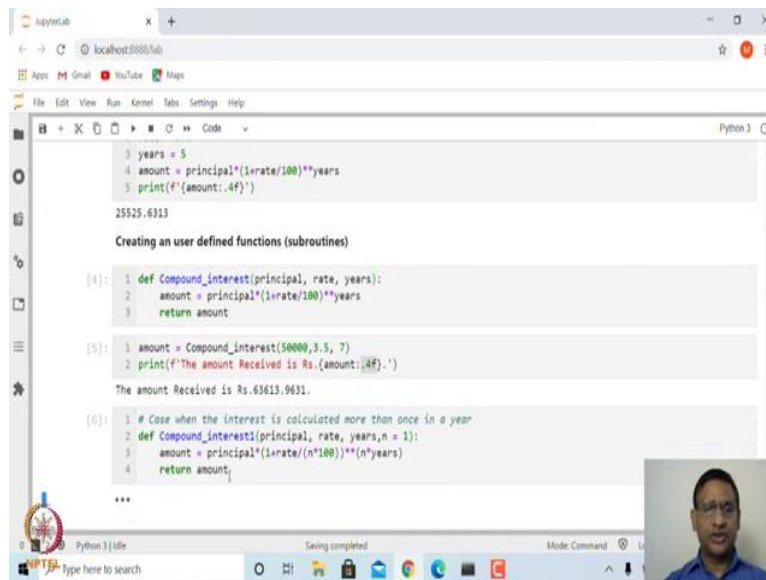
So, I have just copied this particular line inside this and then, what it should return? The function should return some value. It is returning the amount after so many years of investment. Let us run this. One of the important thing is that, you should notice here, there is a colon here and also, whatever comes after this colon, these two lines which is part of the body of the function are indented.

So, indentation is very important in Python. For example, when you type colon and after that if you press enter, automatically your cursor will go inside and all these things will become aligned to this. So, even if I go back and then, hit enter, again, this return amount will be aligned with amount.

Let us run this. Once we have run, how do we call this function? I have to simply say compound interest and input the principal, input the rate and the number of years. So, let us call this function. I will call this as amount equal to compound underscore interest, that is the name of the function and let us say we have invested instead of 20000, suppose we have invested 50000 rupees and at the rate, let us say 3.5 percent and it is invested for 7 years and then, that is the amount you will get and you want to print what will be the amount received after 7 years. So, this is 63613.96331.

Of course, here we are printing only 4 decimal places, you can print more decimal places or you can print significant number of digits, all these possibilities are there.

(Refer Slide Time: 04:25)



The screenshot shows a JupyterLab window with a Python 3 kernel. The code in the cell is as follows:

```
years = 5
amount = principal*(1+rate/100)**years
print(f'{amount:.4f}')

25525.6313

Creating a user defined functions (subroutines)

[4]: 1 def Compound_interest(principal, rate, years):
      2     amount = principal*(1+rate/100)**years
      3     return amount

[5]: 1 amount = Compound_interest(50000,3.5, 7)
      2 print(f'The amount Received is Rs.{amount:.4f}.')

The amount Received is Rs.63613.9631.

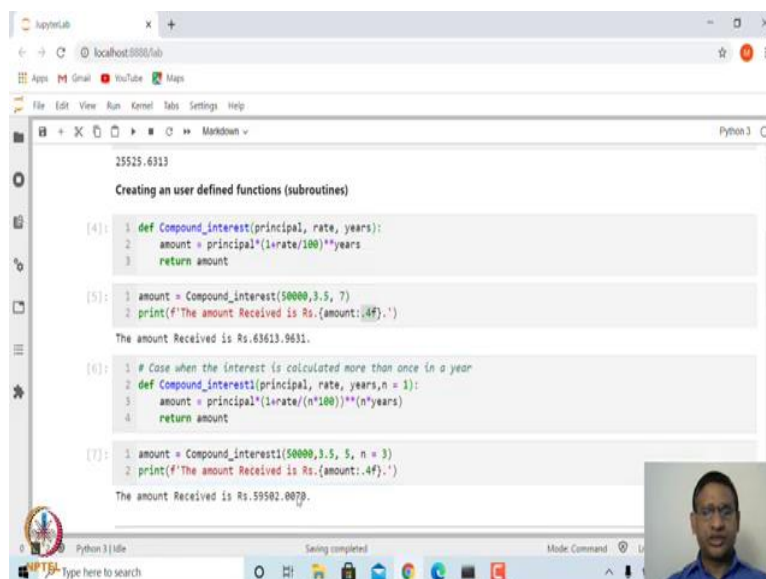
[6]: 1 # Case when the interest is calculated more than once in a year
      2 def Compound_interest1(principal, rate, years,n = 1):
      3     amount = principal*(1+rate/(n*100))**(n*years)
      4     return amount
      ...
```

The output of the first cell is 25525.6313. The output of the second cell is The amount Received is Rs.63613.9631. The output of the third cell is The amount Received is Rs.59582.0079.

Now, suppose you want this compound interest to be calculated not annually, but it is calculated quarterly or it is calculated every month etc.

So in that case the number of times compound interest calculated per year, we can give this as another argument n equal to 1, this is a default parameter. So, even if you do not give this, it will take default value as n equal to 1; but in case you give this value, it will calculate. So, that is the only change in the previous user defined function. Let me call this compound interest1, and let us run this and again. Let us call this function.

(Refer Slide Time: 05:28)



The screenshot shows a JupyterLab window with a Python 3 kernel. The code in the cell is as follows:

```
25525.6313

Creating a user defined functions (subroutines)

[4]: 1 def Compound_interest(principal, rate, years):
      2     amount = principal*(1+rate/100)**years
      3     return amount

[5]: 1 amount = Compound_interest(50000,3.5, 7)
      2 print(f'The amount Received is Rs.{amount:.4f}.')

The amount Received is Rs.63613.9631.

[6]: 1 # Case when the interest is calculated more than once in a year
      2 def Compound_interest1(principal, rate, years,n = 1):
      3     amount = principal*(1+rate/(n*100))**(n*years)
      4     return amount

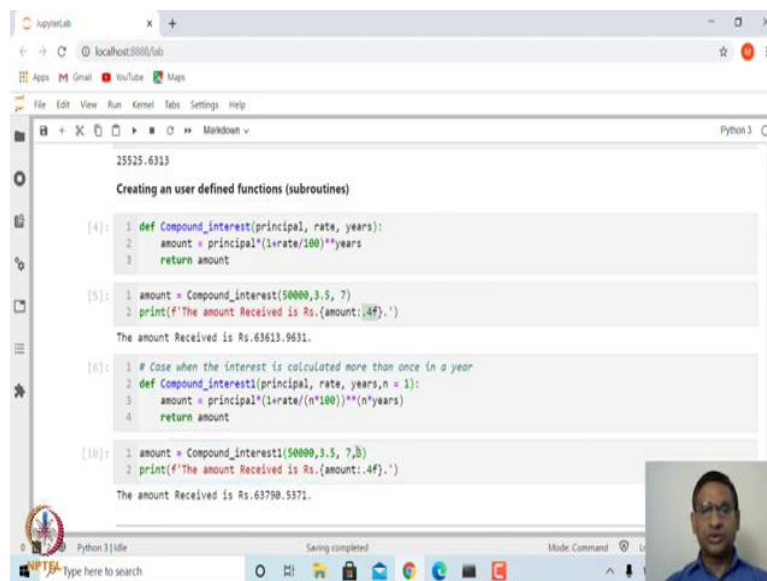
[7]: 1 amount = Compound_interest1(50000,3.5, 5, n = 3)
      2 print(f'The amount Received is Rs.{amount:.4f}.')

The amount Received is Rs.59582.0079.
```

The output of the first cell is 25525.6313. The output of the second cell is The amount Received is Rs.63613.9631. The output of the third cell is The amount Received is Rs.59582.0079.

When we call this function, I have to say compound interest1 and let us say the investment is made 50000 and again this is 3.5 and for 5 years, the number of times it is calculated is 3. That means, every quarter, the interest is calculated and then, let us print what are the amount we get. Again you can see here this amount is 59502. Earlier, it was 7 years; so let us make it 7 years and then, see what is the amount. So, you can see that amount in this case is more than the previous amount, when the interest was calculated annually.

(Refer Slide Time: 06:17)



```
25525.6313

Creating an user defined functions (subroutines)

[4]: 1 def Compound_interest(principal, rate, years):
      2     amount = principal*(1+rate/100)**years
      3     return amount

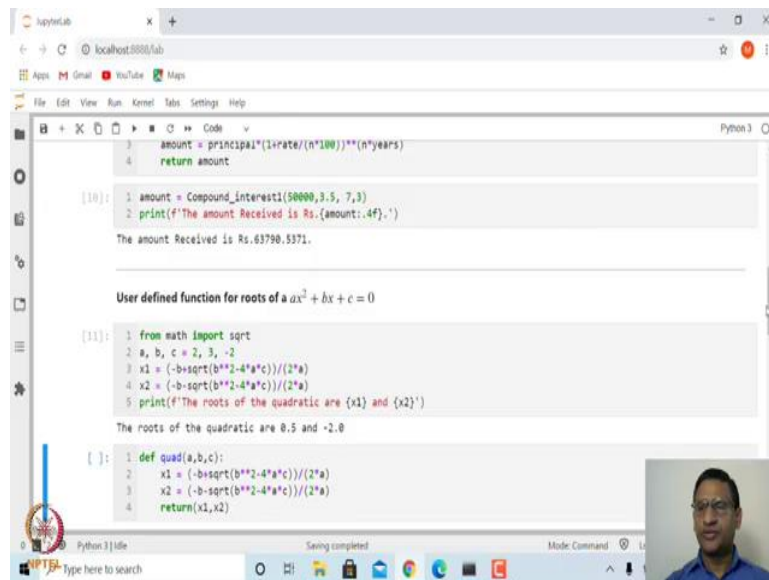
[5]: 1 amount = Compound_interest(50000,3.5, 7)
      2 print(f'The amount Received is Rs.{amount:.4f}.')
The amount Received is Rs.63613.9631.

[6]: 1 # Case when the interest is calculated more than once (n a year)
      2 def Compound_interest1(principal, rate, years,n = 1):
      3     amount = principal*(1+rate/(n*100))**(n*years)
      4     return amount

[10]: 1 amount = Compound_interest1(50000,3.5, 7,3)
       2 print(f'The amount Received is Rs.{amount:.4f}.')
The amount Received is Rs.63798.5371.
```

I have not given this n equal to 3; if I do not give this n equal to 3, then what will happen? Still it will calculate, but it will take the default value of n as 1. Also, you need not say n equal to, 1 you can simply say this is 3; then again, it will calculate this. So, you need not always say n equal to 3. This is how you define a user defined function. Sometimes you give some default value to the argument.

(Refer Slide Time: 07:03)



The screenshot shows a JupyterLab window with a browser at localhost:8888/lab. The code editor contains three Python cells. The first cell defines a function to calculate compound interest. The second cell calls this function with parameters 50000, 3.5, and 7, printing the result. The third cell defines a function to find the roots of a quadratic equation $ax^2 + bx + c = 0$ using the quadratic formula. It imports the sqrt function from the math module, calculates the discriminant, and returns the two roots. The output of the third cell shows the roots 0.5 and -2.0. A small video inset of a man is visible in the bottom right corner of the JupyterLab window.

```
amount = principal*(1+rate/(n*100))**(n*years)
return amount

[10]: 1 amount = Compound_Interest(50000,3.5, 7,3)
2 print(f'The amount Received is Rs.{amount:.4f}.')
The amount Received is Rs.63790.5371.

User defined function for roots of a  $ax^2 + bx + c = 0$ 

[11]: 1 from math import sqrt
2 a, b, c = 2, 3, -2
3 x1 = (-b+sqrt(b**2-4*a*c))/(2*a)
4 x2 = (-b-sqrt(b**2-4*a*c))/(2*a)
5 print(f'The roots of the quadratic are {x1} and {x2}')
The roots of the quadratic are 0.5 and -2.0

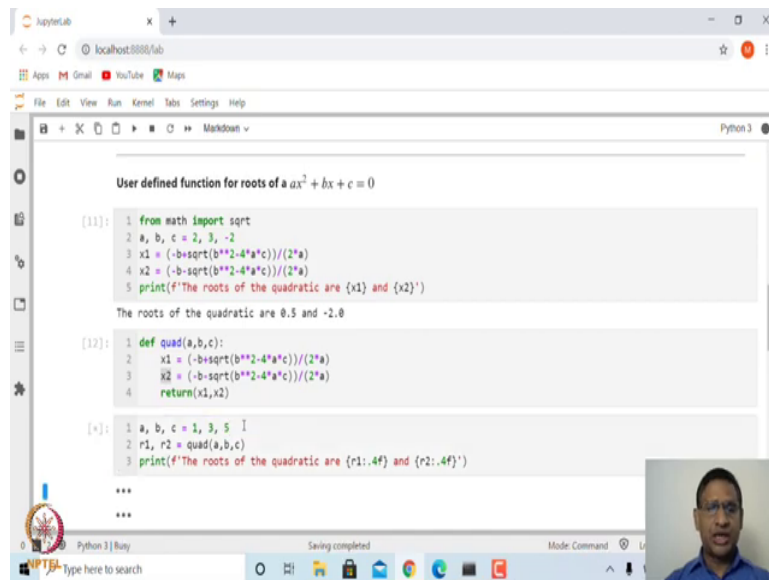
[ ]: 1 def quad(a,b,c):
2     x1 = (-b+sqrt(b**2-4*a*c))/(2*a)
3     x2 = (-b-sqrt(b**2-4*a*c))/(2*a)
4     return(x1,x2)
```

Now, let us look at another example. This again, we have done earlier finding roots of a quadratic $ax^2 + bx + c = 0$. We want to make a user defined function for this. So, first let us make a user defined function, when we know that the roots exist and the roots, we know that it is given by $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, in case the roots are real. This is what we have done. This is what we already have done. I have just copied this from the earlier the class.

We are first importing this square root from math module and suppose a, b, c is equal to 2, 3, minus 2 respectively. So, a is 2, b is 3 and c is minus 2. So, we are looking at $2x^2 + 3x - 2 = 0$ and the roots are x_1 is equal to $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$ and x_2 is $\frac{-b - \sqrt{b^2 - 4ac}}{2a}$ and then, we can print what are the roots.

So, let us execute this. Now, we want to make user defined function out of this. Then what will be the inputs in this case or the arguments? It will be a, b and c. So, you input a, b and c and return the values x_1 and x_2 .

(Refer Slide Time: 08:57)



```
User defined function for roots of  $ax^2 + bx + c = 0$ 

[11]: 1 from math import sqrt
      2 a, b, c = 2, 3, -2
      3 x1 = (-b+sqrt(b**2-4*a*c))/(2*a)
      4 x2 = (-b-sqrt(b**2-4*a*c))/(2*a)
      5 print(f'The roots of the quadratic are {x1} and {x2}')

The roots of the quadratic are 0.5 and -2.0

[12]: 1 def quad(a,b,c):
      2     x1 = (-b+sqrt(b**2-4*a*c))/(2*a)
      3     x2 = (-b-sqrt(b**2-4*a*c))/(2*a)
      4     return(x1,x2)

[*]: 1 a, b, c = 1, 3, 5
      2 r1, r2 = quad(a,b,c)
      3 print(f'The roots of the quadratic are {r1:.4f} and {r2:.4f}')

***
***
```

Now, so, let us do that. We are calling this function as quad; and the arguments are a, b, c and they calculate x1 is equal to this, x2 equal to this and then, returns x1 x2. So, in this, the two returns are x1 and x2.

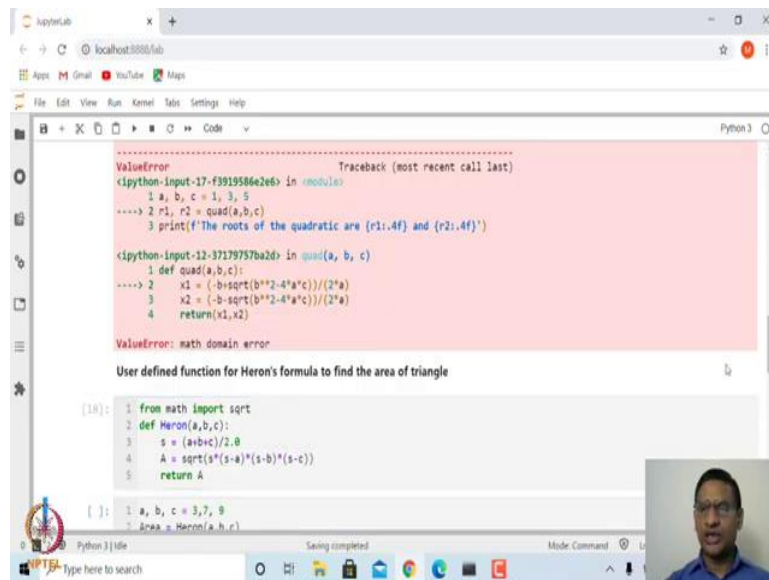
Let us run this and let us call this function quad. When we call this function, this is the returns which are two things; x1 and x2, both the roots. I will store then in x1 and x2. So, first root will be stored in x1, second root will be stored in x2. You may be wondering here I have given x1 and here, I have given x2.

So, this is same thing x1, x2, I have to show. Note that is not necessary; I can even say r1 comma r2; and in this case, we will change this to r1 and this is r2. Now, let us run this. You will get the roots of the quadratic are 0.5 and minus 2.0. Let us change the value. Suppose, instead of 2 here, I will make it 1 and then, the roots are this.

Of course, it is printing so many decimal places. You can put here for example, colon dot 4 f and here also colon dot 4 f. Then, it will print only 4 decimal places.

What happens in case, the roots are imaginary? So, here this x1, x2 are calculated, when the roots are real, that means, discriminant is non-negative. Now, suppose, let us say instead of this I make it here 5 and then, when I run this, it will say domain error.

(Refer Slide Time: 10:43)



The screenshot shows a JupyterLab window with a Python 3 kernel. The top pane displays a traceback for a `ValueError: math domain error` that occurred in a function named `quad(a, b, c)`. The function code is as follows:

```
def quad(a, b, c):  
    1 x1 = (-b+sqrt(b**2-4*a*c))/(2*a)  
    2 x2 = (-b-sqrt(b**2-4*a*c))/(2*a)  
    3 return(x1,x2)
```

The bottom pane shows a user-defined function for Heron's formula, `Heron(a, b, c)`, which imports `sqrt` from the `math` module and calculates the area of a triangle given its sides `a`, `b`, and `c`.

```
[10]: 1 from math import sqrt  
      2 def Heron(a,b,c):  
      3     s = (a+b+c)/2.0  
      4     A = sqrt(s*(s-a)*(s-b)*(s-c))  
      5     return A
```

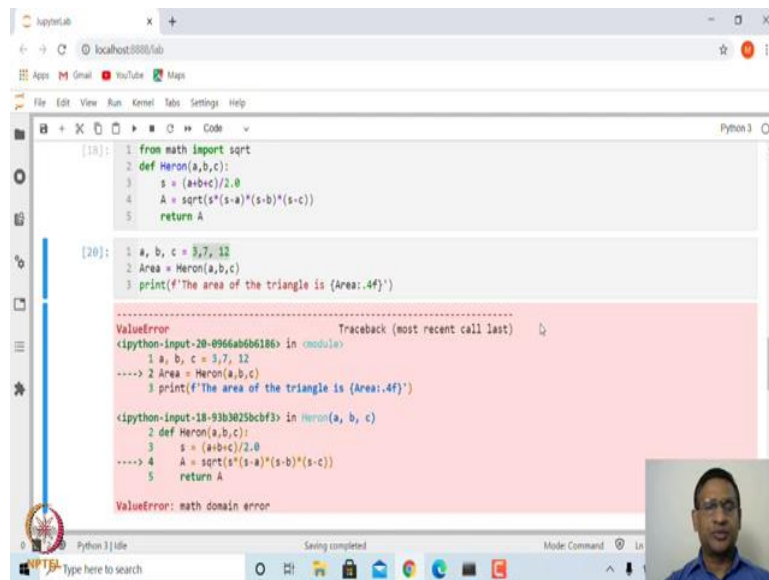
Below the function definition, a code cell shows the function being called with `a=3, b=7, c=9` and the result stored in `Area`.

```
[ ]: 1 a, b, c = 3,7, 9  
      2 Area = Heron(a,b,c)
```

You can see here math domain error because in this case the discriminant is negative and which is not calculated. So, that is not taken care in this case. Later on, we will we look at all possible cases and will make a user define function out of this.

Similarly, suppose we want to make a user defined function for calculating the area of a triangle using Heron's formula. Since Heron's formula will require calculation of a square root, let us import square root from math module and then, the name of the function is Heron, the arguments are a, b, c. Calculate s is equal to semi perimeter and the area is A and then, return the area. So, this is fairly simple.

(Refer Slide Time: 11:35)



```
1 from math import sqrt
2 def Heron(a,b,c):
3     s = (a+b+c)/2.0
4     A = sqrt(s*(s-a)*(s-b)*(s-c))
5     return A

[20]: 1 a, b, c = 3,7, 12
2 Area = Heron(a,b,c)
3 print(f'The area of the triangle is (Area:.4f)')
```

ValueError: math domain error

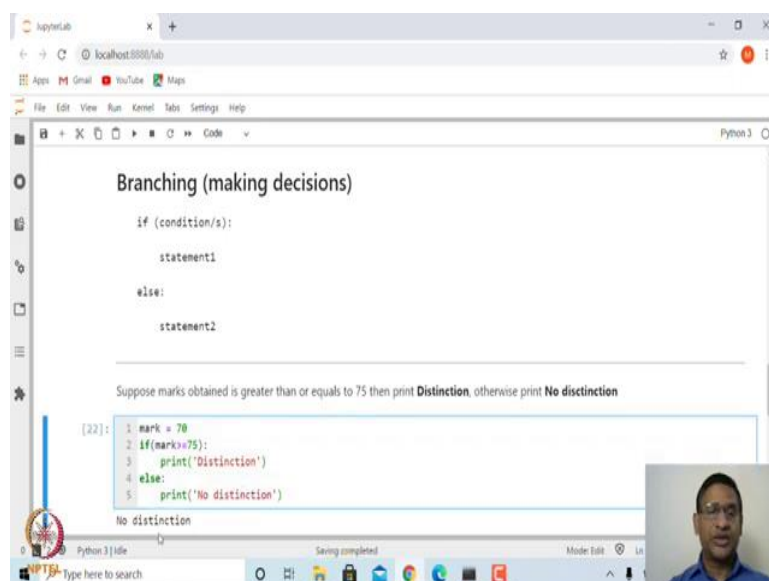
Traceback (most recent call last):

```
<ipython-input-20-0966ab6b6186> in <module>
1 a, b, c = 3,7, 12
----> 2 Area = Heron(a,b,c)
3 print(f'The area of the triangle is (Area:.4f)')
```

```
<ipython-input-18-93b3625bcbf3> in Heron(a, b, c)
2 def Heron(a,b,c):
3     s = (a+b+c)/2.0
----> 4     A = sqrt(s*(s-a)*(s-b)*(s-c))
5     return A
```

And let us again, call this, a, b, c are 3, 7, 9 and the area of this triangle is 68.7856. What if, I give for example, 3, 7 and 12. Then, because this does not form a triangle, there we know that three sides will form triangle and the property should be sum of any two side should be bigger than third side which is not the case in this. So, again, it is giving you math domain error. That is, because this does not form a triangle and one of these things in s minus a, s minus b and, s minus c will become negative and that is why you are getting domain error. So, let us put this back to 8.

(Refer Slide Time: 12:34)



Branching (making decisions)

```
if (condition/s):
    statement1
else:
    statement2
```

Suppose marks obtained is greater than or equals to 75 then print **Distinction**, otherwise print **No distinction**

```
[22]: 1 mark = 70
2 if(mark>=75):
3     print('Distinction')
4 else:
5     print('No distinction')
```

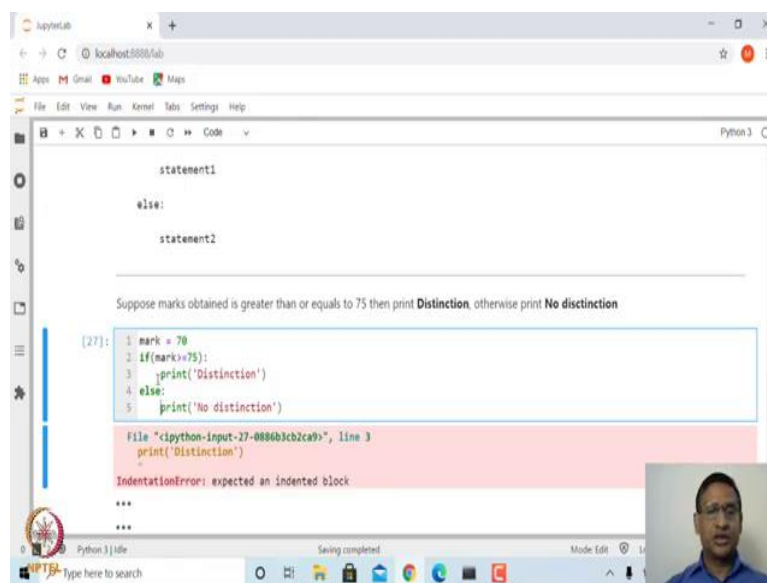
No distinction

Now, for example, in calculating the square root, the roots depends upon whether discriminant is non-negative or negative. Suppose we want to check such cases, then how do we do that? Even in case of a triangle, we want to check whether three sides forms a triangle or not; what kind of triangle it is; all these things, if I want to consider, then we can use what is called 'if' condition; 'if' and 'else' condition.

So, how do we make use of if and else condition? The general syntax for if and else is something like this. if and in the bracket or even without bracket, you write condition. It could be one condition or it could be several conditions. And then these conditions should be logical conditions. So, if you are using equality, it should be logically equal, and things like that, and then, put colon here, again that is important, whatever the expression you want to evaluate or statement you would put it and else, again this this else is aligned with if and then, again put colon. So, in case, these conditions are satisfied, it will evaluate this statement; otherwise, it will evaluate this statement.

So, let us make use of this. Suppose, we want to print distinction in case the marks is greater than equal to 75 and print no distinction otherwise. So, how do I write Python codes for this? So, mark is let us say 70 and in case, if in the bracket marks greater than equal to 75. Then, print distinction and otherwise, that is else, again colon print no distinction.

(Refer Slide Time: 14:48)



The screenshot shows a JupyterLab window with a Python 3 kernel. The code cell contains the following code:

```
statement1  
  
else:  
    statement2
```

Below the code cell, there is a text prompt: "Suppose marks obtained is greater than or equals to 75 then print **Distinction**, otherwise print **No distinction**".

The code cell is executed, and the output shows the following code:

```
[27]: 1 mark = 70  
      2 if(mark>=75):  
      3     print('Distinction')  
      4 else:  
      5     print('No distinction')
```

Below the code, there is a red error message: "File <ipython-input-27-0884b3cb2cab>, line 3
print('Distinction')
IndentationError: expected an indented block".

So, if I run this, since the marks is 70 which is less than 75. So, it will go to this particular branch and it will print no distinction. If I have marks, let us say 78 and then, if I run this, it

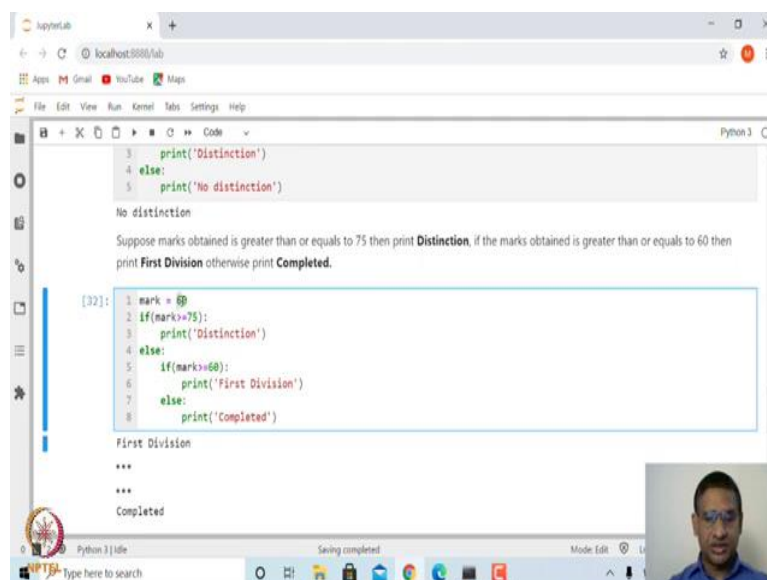
will print distinction. So, this branch is taken into consideration. Of course, this if condition has existence without else also.

Suppose if I cut this, and just run this one. So, it will still work. Let us say instead of 78, if I have 70, then it will not print anything because there is only one condition, it is checking if the mark is greater than equal to 75. And this has to be aligned; the alignment has to be proper, the indentation is very important.

And of course, if you if you do not put colon here, then it will give you an error. It gives you an error which says that the syntax is invalid. And if you do not align this print like this; suppose, I put everything in single line, again it will give you an error. It says that expected indented block.

So, that is also important. So, keep all these things in mind, advantage of using this kind of editor is that when I press enter, automatically cursor goes inside and similarly, in case of this, it goes inside.

(Refer Slide Time: 16:16)

A screenshot of a JupyterLab window. The top part shows a code editor with Python code. The code has two sections. The first section is a simple if-else statement. The second section is a more complex if-else statement with nested indentation. The output of the code is displayed below the code editor. The output shows 'No distinction' for the first section and 'First Division' and 'Completed' for the second section. The JupyterLab interface includes a file explorer on the left, a command palette at the bottom, and a small video feed of a person in the bottom right corner.

```
3 print('Distinction')
4 else:
5 print('No distinction')

No distinction

Suppose marks obtained is greater than or equals to 75 then print Distinction, if the marks obtained is greater than or equals to 60 then
print First Division otherwise print Completed.

[32]: 1 mark = 60
      2 if(mark>=75):
      3     print('Distinction')
      4 else:
      5     if(mark>=60):
      6         print('First Division')
      7     else:
      8         print('Completed')

First Division
***
***
Completed
```

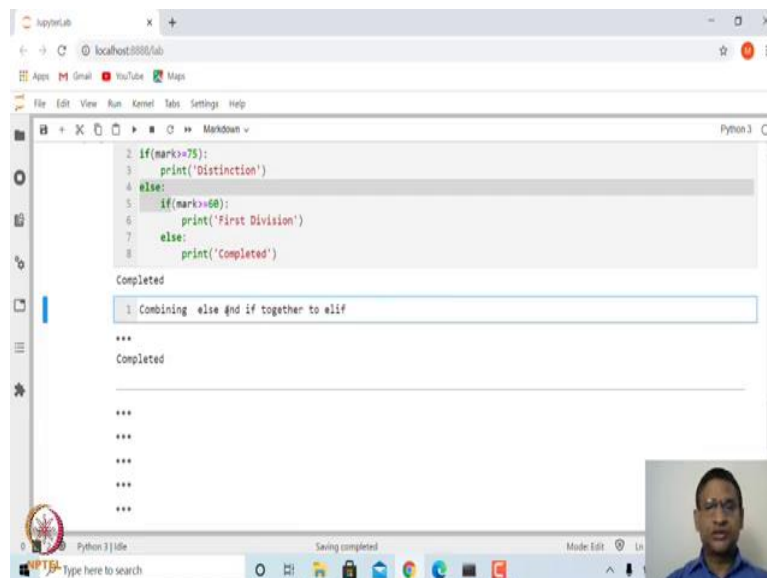
So, even if you put let us say this, if you do not aligned properly, it will give you an error again, expected indentation. Also colon is also important.

The colon simply tells you that something is to follow. It is not complete, the some statement has to follow, that is what it means roughly. Now, suppose, I have another condition. For example, if marks is greater than equal to 75, you print distinction. In case, it is greater than equal to 60, print let us say, first division and otherwise, you simply say, the course is completed.

So, how do I say that? In this else, we have to have another if condition. So, you can use what is called 'nested if'. Let us look at that. So, this is how you write. First, put here, this is alright and after that, we are saying if marks is greater than equal 75, then print the first division else. So, this whole thing is inside else; otherwise, you print completed. Now, when you run this, it brings no distinction.. So, it is a first division, if it is 60.

Suppose, the marks is let us say 40 and then, it should print just completed. Whatever is the marks below 60, it will say completed; you can have more condition inside this. Of course, you can also combine this 'if and else' together and you can use elif. Let me show you, how to combine else and if together.

(Refer Slide Time: 18:15)

A screenshot of a JupyterLab window. The top part shows a code editor with the following Python code:

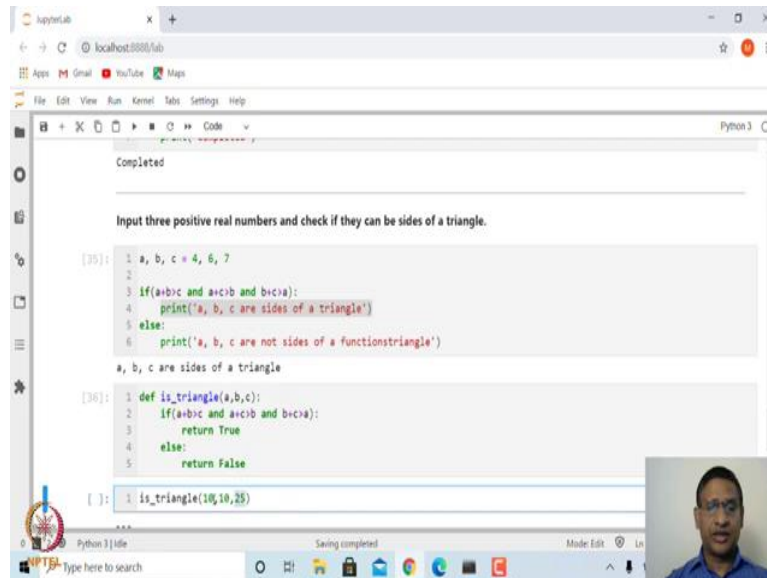
```
1 if(mark>=75):  
2     print('Distinction')  
3 else:  
4     if(mark>=60):  
5         print('First Division')  
6     else:  
7         print('Completed')  
8
```

The output area below the code shows the result of running the code: "Completed". Below the output, there is a text input field containing "Combining else and if together to elif". The bottom part of the screenshot shows a Windows taskbar with various icons and a small video feed of a person in the bottom right corner.

And it should be combining if and, else together to elif. Let me do that. In this case, else and if is combined together in elif and then, marks is greater than equal to 60, then first division

else print completed. So, this is somewhat you are just saving few lines. You can have actually many nestings and you can have so many conditions inside.

(Refer Slide Time: 19:07)



```
Completed

Input three positive real numbers and check if they can be sides of a triangle.

[35]: 1 a, b, c = 4, 6, 7
      2
      3 if(a+b>c and a+c>b and b+c>a):
      4     print('a, b, c are sides of a triangle')
      5 else:
      6     print('a, b, c are not sides of a functiontriangle')

a, b, c are sides of a triangle

[36]: 1 def is_triangle(a,b,c):
      2     if(a+b>c and a+c>b and b+c>a):
      3         return True
      4     else:
      5         return False

[ ]: 1 is_triangle(10,10,25)
```

Now, for example, suppose we input three positive real numbers a, b, c and want to check whether it forms a triangle. We know conditions on a, b, c under which they are sides of a triangle; namely, sum of any two sides should be bigger than the third side. So, how do we do that? In this case, we need to have more than one condition.

So, a, b, c are in this case 4, 6, 6; let me say 4, 6, 7 and if a plus b is greater than c and so, this is the combination; this is adjoining the two conditions by and or you can write or depending upon what you need to use. So, a plus b is greater than c or a and a plus c should be bigger than b and a plus c should be bigger than a.

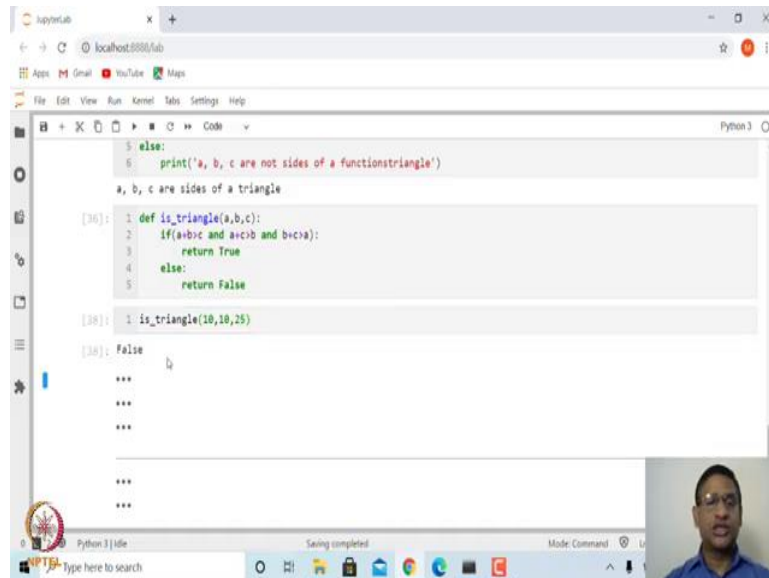
In this case, we want to print a, b, c are sides of a triangle otherwise they are not sides of a triangle. So, here inside if we have used three conditions and they are joined by and. Similarly, suppose you want to make a user defined function out of it; that means, you want to check whether a, b, c forms a triangle or not. So, we can simply copy paste inside this, the function name.

So, the function name, I am giving is underscore triangle and the arguments are a, b, c and then colon. If this condition is satisfied, then print return True; that means, it will be True; otherwise

return False. So, instead of printing something, we are saying return True or False. So, let us run this and let us call this.

So, if I said triangle 10, 10 and let us say 15. So, this is the answer which is true. It forms a triangle. Instead of 15, if I say 25, then a plus b is not greater than c; so, it should say False.

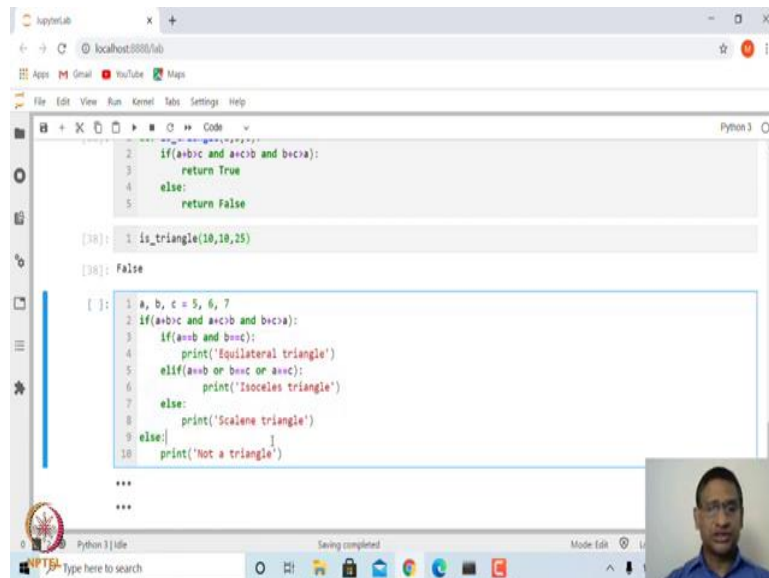
(Refer Slide Time: 21:22)



Suppose, instead of just printing whether it forms a triangle or not; in case it forms a triangle, we want to check whether it is an equilateral triangle or isosceles triangle or scalene triangle. In that case, we can say check if a plus b is greater than c and this. So, this is the first condition to check if it is a triangle.

So, what we are doing? In the previous condition, if this is True, instead of True, then we are looking at another if else and we are checking in case a equal to b and b equal to c; then, it will be an Equilateral triangle. Else if if a is equal to b or b is equal to c or a is equal to c, then it will be an Isosceles triangle and otherwise, it will be scalene triangle. And if this condition is not satisfied, then it will not be a triangle.

(Refer Slide Time: 22:18)



The screenshot shows a JupyterLab window with a Python 3 kernel. The code defines a function `is_triangle(a, b, c)` that checks if three sides can form a triangle. It uses a single `if` statement with a combined condition `a+b > c and a+c > b and b+c > a`. If true, it returns `True`; otherwise, it returns `False`. Below the function, there are two calls to `is_triangle(10, 10, 25)`, both returning `False`. A third call `is_triangle(5, 5, 5)` is shown, which would return `True`. The interface includes a file explorer on the left, a top menu bar, and a bottom status bar.

```
def is_triangle(a, b, c):
    if(a+b > c and a+c > b and b+c > a):
        return True
    else:
        return False

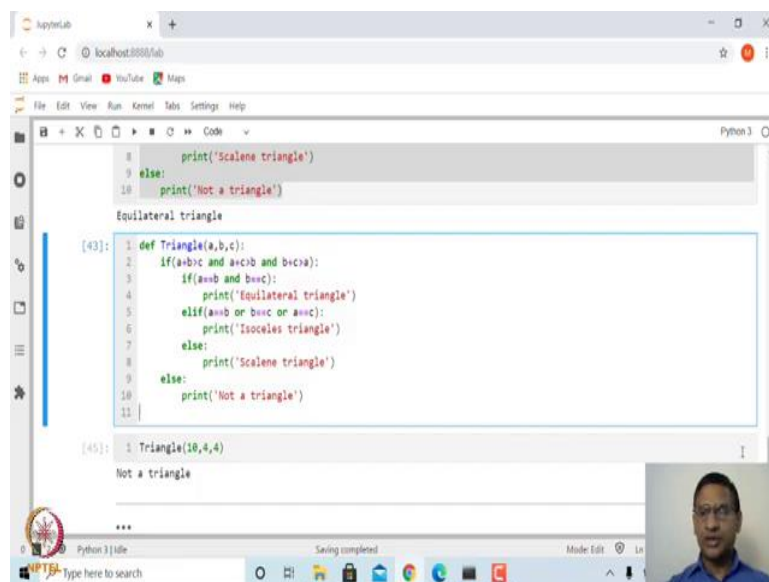
[39]: is_triangle(10,10,25)

[39]: False

[40]: is_triangle(5,5,5)
```

So, that is why this else is aligned with the first if and this else are inside the first block, first if condition. Let us execute this. In this case and this is scalene triangle; if I say here 5 and then run this it is an isosceles triangle. Now suppose I have put this also as 5, then this is an equilateral triangle. Again, you can make a user defined function out of this.

(Refer Slide Time: 22:56)



The screenshot shows a JupyterLab window with a Python 3 kernel. The code defines a function `Triangle(a,b,c)` that checks if three sides can form a triangle. It uses a single `if` statement with a combined condition `a+b > c and a+c > b and b+c > a`. If true, it prints 'Equilateral triangle'; if false, it prints 'Not a triangle'. Below the function, there is a call to `Triangle(10,4,4)`, which outputs 'Not a triangle'. The interface includes a file explorer on the left, a top menu bar, and a bottom status bar.

```
def Triangle(a,b,c):
    if(a+b > c and a+c > b and b+c > a):
        print('Equilateral triangle')
    else:
        print('Not a triangle')

[43]: Triangle(10,4,4)

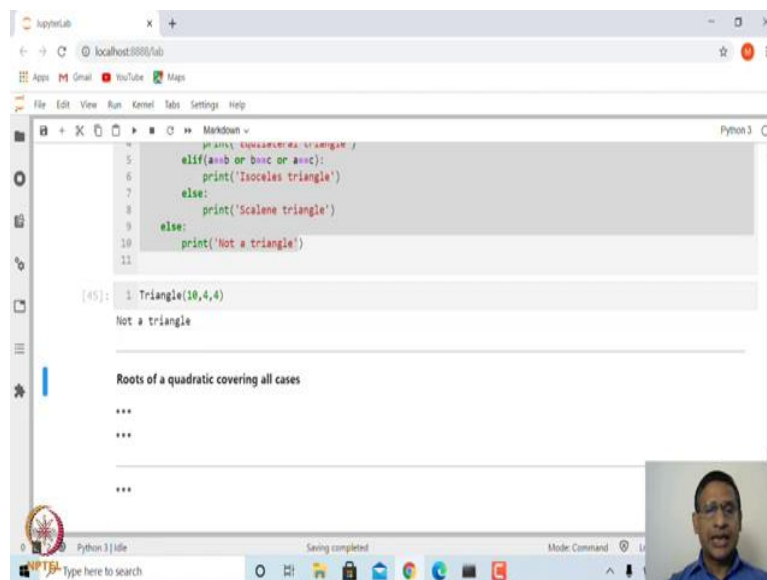
[43]: Not a triangle
```

So, let me call this as triangle. So, all I am doing is that whatever we have done here, we had just copied inside this function called triangle and then, a, b, c colon and then, this these things are just actually copied inside this.

So, if I run this, I made a triangle function which checks whether a, b, c forms a triangle. If it forms a triangle, it checks whether it is an equilateral triangle or an isosceles triangle or a scalene triangle and if it does not form a triangle, it simply returns not a triangle.

So, if I execute this, it says its an isosceles triangle. If I say for example, 10 in this case; then, it says it is not a triangle. So, you can see here making user defined function is very easy, but actually what you need to do is first of all try to write the body separately and if that runs, then all you need to do is just copy paste this body from the previous one. Also try to split when you have to create a user defined function, try to split the problem into smaller problem and then, try to run.

(Refer Slide Time: 24:22)



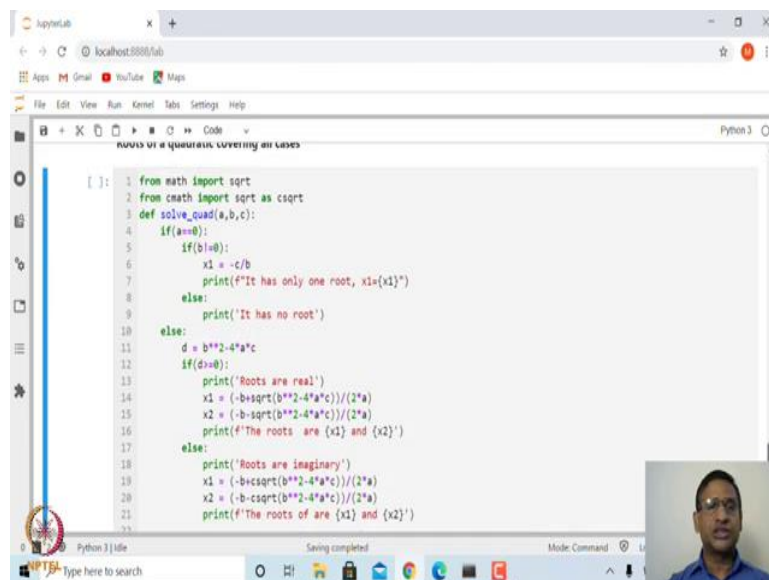
The screenshot shows a JupyterLab window with a Python 3 kernel. The code editor contains the following Python code:

```
def isosceles_triangle(a, b, c):  
    if (a==b or b==c or a==c):  
        print('Isosceles triangle')  
    else:  
        print('Scalene triangle')  
    else:  
        print('Not a triangle')
```

The output area shows the execution of the function `Triangle(10,4,4)`, which results in the output `Not a triangle`. Below the output, there is a section titled "Roots of a quadratic covering all cases" with three empty lines for input.

Now, suppose we want to again write a user defined function for finding roots of a quadratic and we want to cover all the possible cases. That means, look at when the discriminant is non-negative then, print the roots. And if it is negative, then put print the roots. So, how do we do that?

(Refer Slide Time: 24:48)



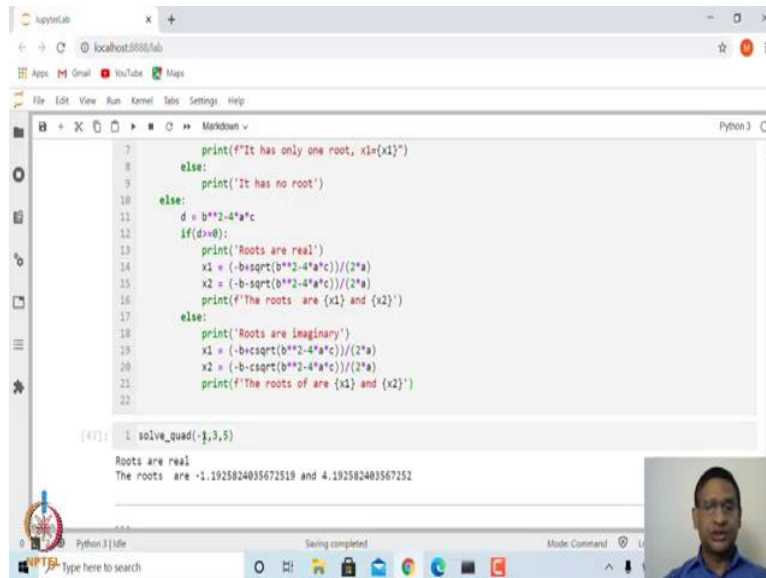
```
1 from math import sqrt
2 from cmath import sqrt as csqrt
3 def solve_quad(a,b,c):
4     if(a==0):
5         if(b!=0):
6             x1 = -c/b
7             print(f'It has only one root, x1={x1}')
8         else:
9             print('It has no root')
10    else:
11        d = b**2-4*a*c
12        if(d>=0):
13            print('Roots are real')
14            x1 = (-b+sqrt(b**2-4*a*c))/(2*a)
15            x2 = (-b-sqrt(b**2-4*a*c))/(2*a)
16            print(f'The roots are {x1} and {x2}')
17        else:
18            print('Roots are imaginary')
19            x1 = (-b+csqrt(b**2-4*a*c))/(2*a)
20            x2 = (-b-csqrt(b**2-4*a*c))/(2*a)
21            print(f'The roots of are {x1} and {x2}')
```

So, in this case again, we have to use if and else. So, two things I am importing, one is square root from math module and in case the discriminant is negative, then the roots will be complex and that case we can compute the complex roots.

So, this square root function from cmath, it is imported as csqrt. So, let us again define user defined function, solve underscore quadratic is the name and a, b, c are the arguments. So, first we will check if a is 0, then we will check whether b is 0. If b is non zero, the root will have only one root that is minus c upon b and if b is also 0, then you get 0; that means, you have just a constant equal to something which is not a quadratic, you cannot do anything.

So, in that case, we are printing that it has no root and so, if this is not the case. If a is equal to 0 is not true, that means, a is non zero, in that case we will check the discriminant. This is the discriminant. And then, once you have found the discriminant, you check whether discriminant is non-negative. If discriminant is non-negative, then print the roots are real and then, compute the roots x1, x2 and then print the two roots. So, here we are not using return, we are using print.

(Refer Slide Time: 26:11)



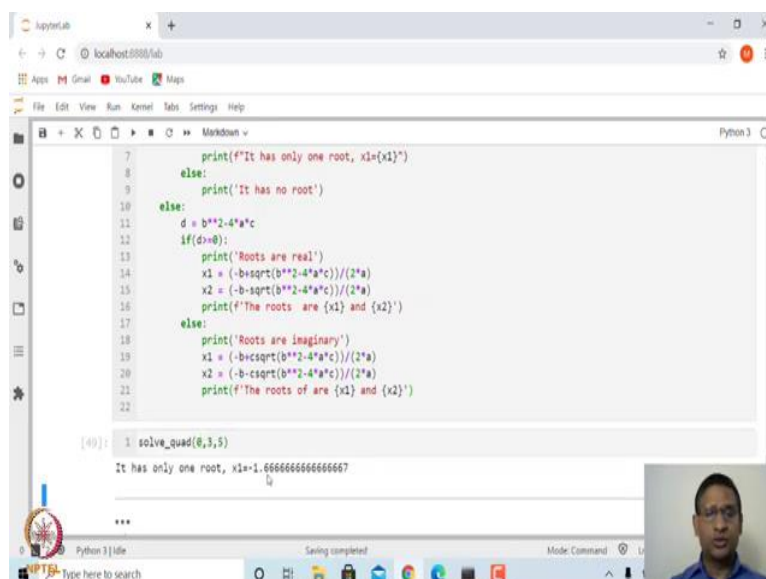
```
7 print("It has only one root, x1={x1}")
8 else:
9     print('It has no root')
10 else:
11     d = b**2-4*a*c
12     if(d>=0):
13         print('Roots are real')
14         x1 = (-b+sqrt(b**2-4*a*c))/(2*a)
15         x2 = (-b-sqrt(b**2-4*a*c))/(2*a)
16         print(f'The roots are {x1} and {x2}')
17     else:
18         print('Roots are imaginary')
19         x1 = (-b+csqrt(b**2-4*a*c))/(2*a)
20         x2 = (-b-csqrt(b**2-4*a*c))/(2*a)
21         print(f'The roots of are {x1} and {x2}')
22
```

```
[47]: 1 solve_quad(-1,3,5)
Roots are real
The roots are -1.1925824035672519 and 4.192582403567252
```

You can also use return. Otherwise, if discriminant negative, is this else is true so, otherwise the roots are imaginary; that means, if this is not true means discriminant is less than 0. So, the roots are imaginary and then, print the two roots x1, x2 using csqrt, that means, it is taking square root from complex math.

So, that is the function. It is fairly simple. Let us again run this and call this function. In this case, suppose I say solve x square plus 3 x plus 5 equal to 0, the roots are imaginary and it is printing the imaginary roots.

(Refer Slide Time: 26:57)

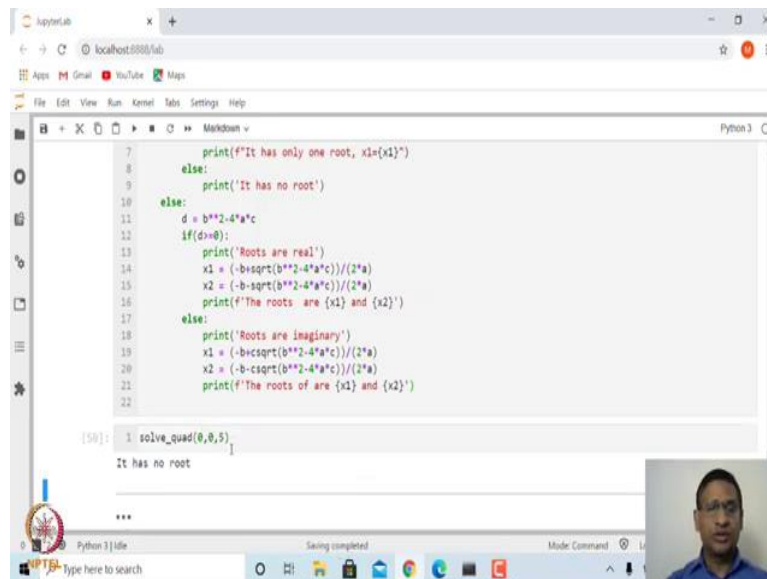


```
7 print("It has only one root, x1={x1}")
8 else:
9     print('It has no root')
10 else:
11     d = b**2-4*a*c
12     if(d>=0):
13         print('Roots are real')
14         x1 = (-b+sqrt(b**2-4*a*c))/(2*a)
15         x2 = (-b-sqrt(b**2-4*a*c))/(2*a)
16         print(f'The roots are {x1} and {x2}')
17     else:
18         print('Roots are imaginary')
19         x1 = (-b+csqrt(b**2-4*a*c))/(2*a)
20         x2 = (-b-csqrt(b**2-4*a*c))/(2*a)
21         print(f'The roots of are {x1} and {x2}')
22
```

```
[49]: 1 solve_quad(0,3,5)
It has only one root, x1=1.6666666666666667
***
```

If I put here 0, it will have only one root that is what it is printing.

(Refer Slide Time: 27:02)



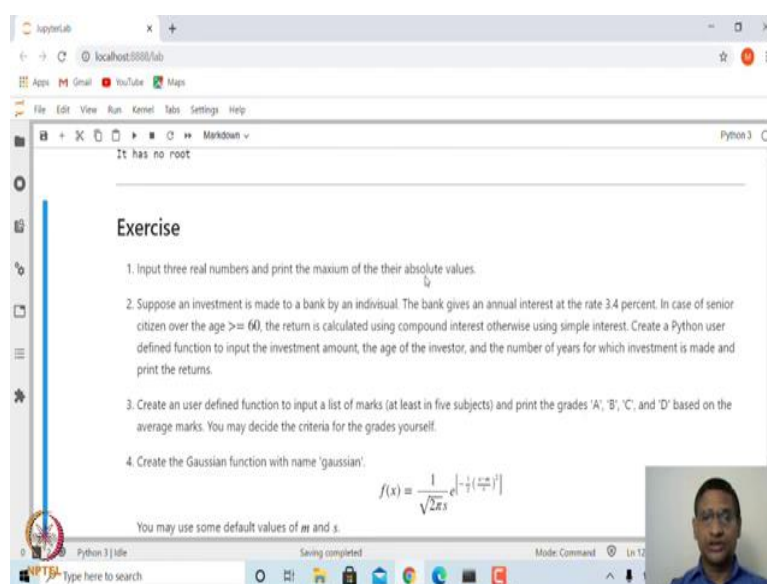
```
7 print(f'It has only one root, x1={x1}')
8 else:
9     print('It has no root')
10 else:
11     d = b**2-4*a*c
12     if(d==0):
13         print('Roots are real')
14         x1 = (-b+sqrt(b**2-4*a*c))/(2*a)
15         x2 = (-b-sqrt(b**2-4*a*c))/(2*a)
16         print(f'The roots are {x1} and {x2}')
17     else:
18         print('Roots are imaginary')
19         x1 = (-b+csqrt(b**2-4*a*c))/(2*a)
20         x2 = (-b-csqrt(b**2-4*a*c))/(2*a)
21         print(f'The roots of are {x1} and {x2}')
22
```

[50]: 1 solve_quad(0,0,5)

It has no root

If this is also 0, it will give me the error that it has no root. So, that is how you can make use of this function. You could also try to use return instead of print. So, you may just try to change this wherever you have print, you can change it to return. Of course, in this case, here there is no return. You have to have print; but here x1, x2 you can return. So, I expect you to do that exercise.

(Refer Slide Time: 27:49)



Exercise

1. Input three real numbers and print the maximum of their absolute values.
2. Suppose an investment is made to a bank by an individual. The bank gives an annual interest at the rate 3.4 percent. In case of senior citizen over the age ≥ 60 , the return is calculated using compound interest otherwise using simple interest. Create a Python user defined function to input the investment amount, the age of the investor, and the number of years for which investment is made and print the returns.
3. Create a user defined function to input a list of marks (at least in five subjects) and print the grades 'A', 'B', 'C', and 'D' based on the average marks. You may decide the criteria for the grades yourself.
4. Create the Gaussian function with name 'gaussian'.

$$f(x) = \frac{1}{\sqrt{2\pi s}} e^{-\frac{1}{2} \left(\frac{x-m}{s} \right)^2}$$

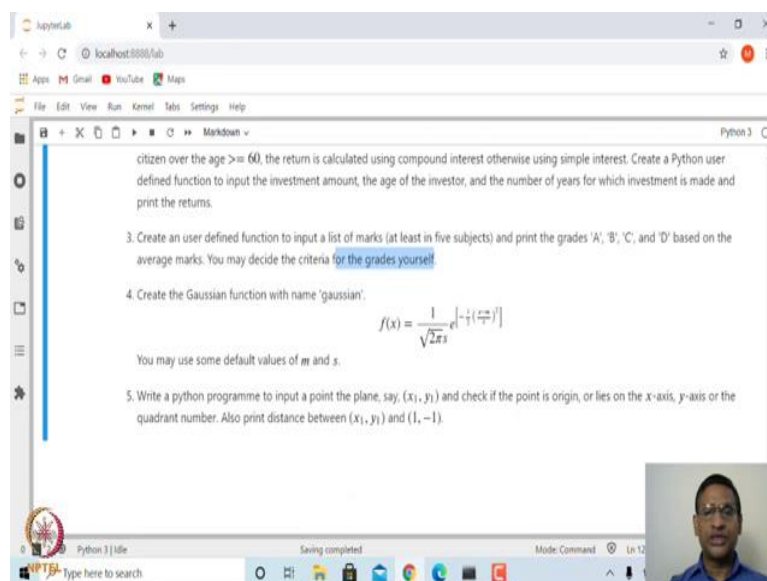
You may use some default values of m and s .

Now, before I end, let me leave you with some simple exercises which will make use of creating user defined function based on certain conditions. So, these are the few exercises.

The first exercise is input 3 real numbers and print the maximum of their absolute values. The absolute value you can get from math module. The next exercise is input an investment and calculate the the return; the bank gives return at 3.4 percent, but that depends upon whether the person is senior citizen or not a senior citizen. In case of senior citizen, the return is through compound interest; otherwise, simple interest.

And then, third one is to create an user defined function to input a list of marks. So, you know how to create a list. So, your argument will be a list and then, it should calculate what is the number of elements in the list and based on that, it should calculate the average and based on the average, you can assign grades 'A', 'B', 'C', 'D'. You can have your own criteria of grading.

(Refer Slide Time: 28:48)



And the fourth one is to create a user defined function for this Gaussian function. This is $f(x)$ equal to $\frac{1}{\sqrt{2\pi}s}$ times e to the power minus half x minus m by s to the whole square and m and s , you can take some default value. For example, you can take m equal to 0 and s is equal to 1, that is actually standard normal distribution function.

And the fifth problem is input a point, you can input as order pair or you can input for example, as a tuple and check if that point is origin. If it is lying on the x axis or y axis, you can print that and otherwise you can print in which quadrant it lies and also, print the distance between that point and 1 comma minus 1. These are the few exercises.

Thank you very much. I will see you in the next lecture. You should solve these exercises. Of course, we will also upload the solution of these problems, they are fairly simple.

Thank you very much.