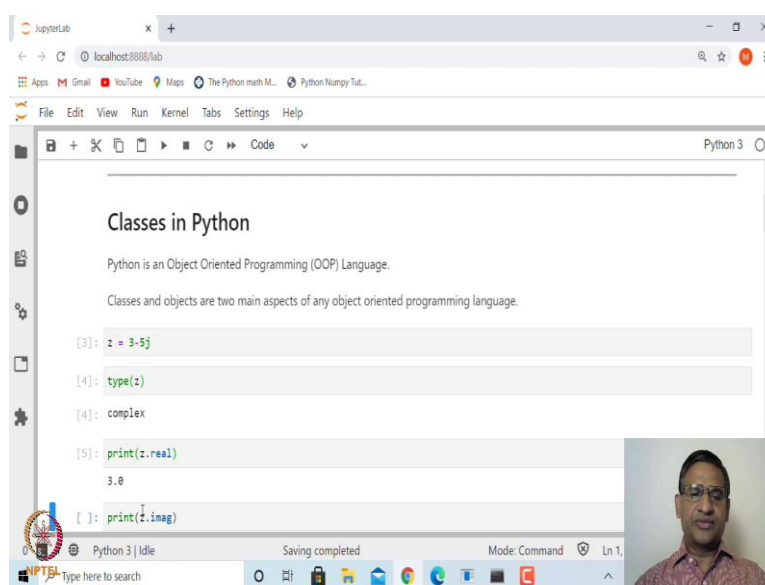


**Computational Mathematics with SageMath**  
**Prof. Ajit Kumar**  
**Department of Mathematics**  
**Institute of Chemical Technology, Mumbai**

**Lecture - 13**  
**Classes in Python – Part 01**

Welcome to the 12th lecture on Computational Mathematics with SageMath. In this lecture, we are going to learn about Classes in Python very briefly. So, let us get started.

(Refer Slide Time: 00:28)



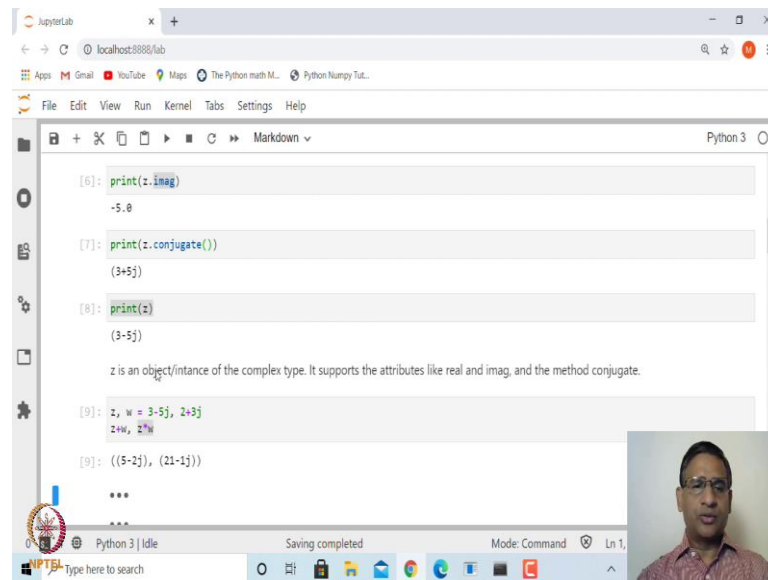
```
[3]: z = 3-5j
[4]: type(z)
[4]: complex
[5]: print(z.real)
3.0
[ ]: print(z.imag)
```

As you know, Python is an object-oriented programming language, and classes and objects are two main aspects of any object-oriented programming language. So, we will see how to create classes, and what are the objects.

Let us look at, suppose we create  $z$ , a complex number, which is  $3 - 5i$ . Once we create this in Python, so  $z$  is a variable. Actually, it is an object of some class. So, if we look at what is type of  $z$ , it says complex, so that means,  $z$  is an object from the class called complex.

And we have seen, we can apply some methods to this  $z$ . So, for example, if I ask it to look at  $z$  dot real, then we will get value, the real value of  $z$ , which is 3.

(Refer Slide Time: 01:53)



```
[6]: print(z.imag)
-5.8

[7]: print(z.conjugate())
(3+5j)

[8]: print(z)
(3-5j)

z is an object/instance of the complex type. It supports the attributes like real and imag, and the method conjugate.

[9]: z, w = 3-5j, 2+3j
z+w, z*w

[9]: ((5-2j), (21-1j))

...
```

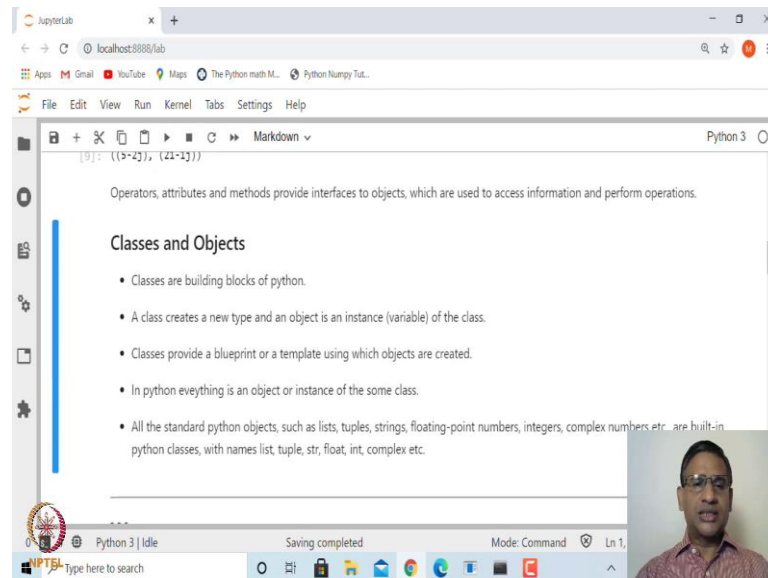
Similarly, if I say `z.imag`, imaginary part, this will give me minus 5. So, real part, and imaginary part we are able to find out from this `z`, and using attribute `real` and `imag`. Similarly, you can, you can look at `z.conjugate`, and open round bracket, it will give you the conjugate of `z`, which is 3 plus 5 j.

So, in this case, already you can see here `z` when we have created this object `z`, some attributes we can say variables, and methods you are able to apply on `z`. So, this is already an example of a class in which you have some attributes, and methods, ok? So, in general, how do we create these type of classes on our own? This is what we are going to study.

So, if I even, if I, if I say, `print z`, it tells you what is this `z`. So, where is this print coming from? From this class, this again we will see. So, in this case, actually what we have done? We have created `z` as an object or instance of the type `complex`, which is also known as class, and it supports attributes like `real`, and `imag`, that is what we saw, and a method in this case, known as `conjugate`.

So, in this case, you, you can just notice when we have said `z.imag`, there is no empty round bracket, whereas, `z.conjugate` ends with empty round bracket, that is because `conjugate` is a method, whereas, this is just an attribute to this class. So, we will, we will see more details on this.

So, even if you ask, if you define one more complex number, let us say  $z$ ,  $w$ , which is  $2$  plus  $3i$ , and  $z$  is  $3$  minus  $5j$ , then you can add  $z$  and  $w$ , you can multiply  $z$  and  $w$ , ok? So, again these, these operations are permitted when you create, or these operations are permitted inside this complex class, ok? So, all these things we will see in today's class. (Refer Slide Time: 04:48)



So, operators attributes like `real`, `imag`, and methods like `conjugate`, these actually provide interface to an object, in this case, the object  $z$  and  $w$ , and which are, which can be accessed using dot operator that is what we, we have seen, `z.real`, `z.imag`, `z.conjugate` all these things can be accessed through dot operator, ok?

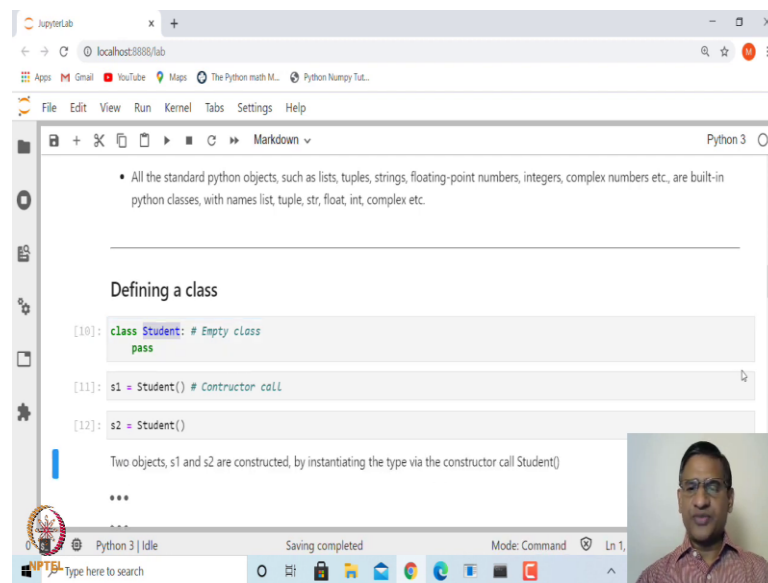
So, now let us look at what are the classes and objects. So, as I said, classes are actually building blocks of Python. Class creates a new type, and an object is an instance or a variable of that particular class. Actually, you can think of class as a blueprint or a template using which you create objects. One can think of class as a factory of objects.

So, several objects you can create with the same class or same type. In fact, in Python, everything is an object or instance, and we have seen several of it. For example, we have seen lists, we have seen tuples, we have seen strings, floating-point numbers, integers, complex numbers, and so on. These are all classes in Python with class name as `list`, `tuple`, `str` for strings, `float` for floating-point numbers, `int`, `complex`, etcetera, ok? So, it is good to know how to create classes, and when you have methods inside these classes how does it look like.

And since SageMath is based on Python programming, and which is also an open-source package, so many times you will be looking at the source code of some function or some methods.

And in that case, you should know what, when you are reading this help document or a source code, you should be able to understand by this knowledge. So, that is the basic idea behind introducing these classes in today's lecture, ok?

(Refer Slide Time: 07:29)



```
[10]: class Student: # Empty class
      pass

[11]: s1 = Student() # Constructor call

[12]: s2 = Student()

Two objects, s1 and s2 are constructed, by instantiating the type via the constructor call Student()

***
```

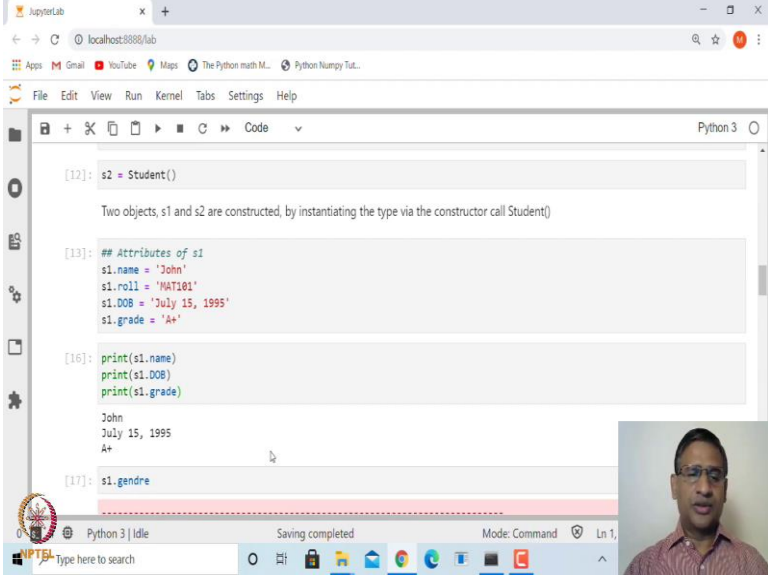
Now, let us see how to create classes. So, to create class, you need to define the keyword class, and followed by the name of the class. So, in this case, I am calling the name as Student, S capital, and then colon, and then if you, the, the simplest class could be a class which is we call as empty class, and you do not write any statement here; you simply write pass, ok? So, this is the simplest of a class you can think of.

Now, what we can do? We can define an object of this class or instance of this class, and how do we define that? So, we will call this as s1, s1 is an object which is, which has student class, or student type. So, we say s1 is equal to student, and empty round bracket. So, let me execute this.

So, we have created s1 as an object of this student class, and you can create many more objects. I can create another student s2, I can create one more s3, s4, or as many as you want, ok?

So, at present s1, s2, are just objects which are student type, student class, but it does not have any, anything inside, ok? It does not have any attribute. As we saw in case of complex number, it has an attribute real, imag, and other things, ok? So, how do we add attributes to any student class? Ok?

(Refer Slide Time: 09:26)



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

```
[12]: s2 = Student()

Two objects, s1 and s2 are constructed, by instantiating the type via the constructor call Student()

[13]: ## Attributes of s1
s1.name = 'John'
s1.roll = 'MAT101'
s1.DOB = 'July 15, 1995'
s1.grade = 'A+'

[16]: print(s1.name)
print(s1.DOB)
print(s1.grade)

John
July 15, 1995
A+

[17]: s1.gendre
```

The output of the code is displayed below the code cell. The first two lines of output are 'John', 'July 15, 1995', and 'A+'. The third line of output is 's1.gendre'. A small video inset of a man is visible in the bottom right corner of the JupyterLab window.

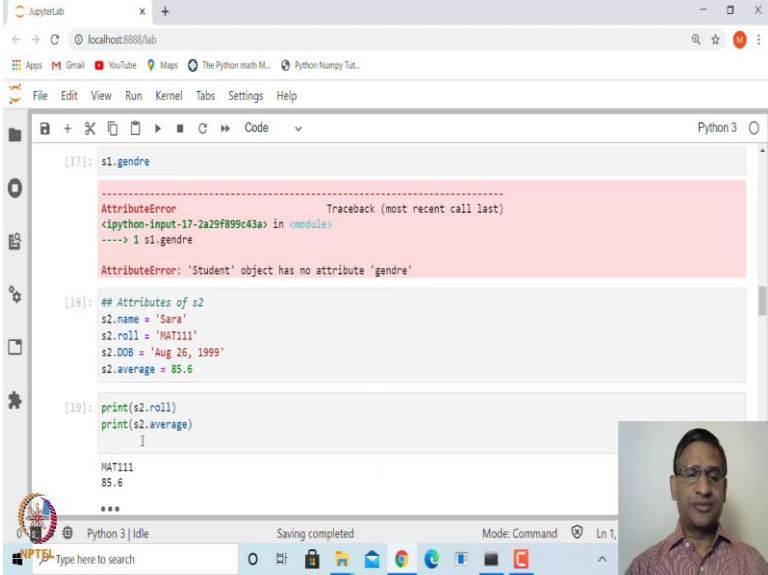
So, in order to add attributes what you need to do? You simply write s1 let us say we want to create an attribute name, name of the student. So, we will say s1 dot name, and then let us say is equal to, the name is John. Similarly, let us say the roll number of s1, how do we create? s1 dot roll, and this roll number is, let us say, MAT 101 or you can have anything.

Similarly, we can create s1 date of birth of s1, as s1 dot DOB, date of birth, and let us say the date of birth is July 1995, 15 July 1995. Similarly, we can create another attribute grade, and it can be created as s1 dot grade. So, let me run all these things. Now, you have created 1, 2, 3, 4 attributes to s1 which are name, roll, roll number, date of birth, and grades, right?

Similarly, you can now, how do we access these, these attributes? So you can simply type s1 dot name, it will give you the name John. If I say s1 dot date of birth, it will give me July 15. If I say s1 print, you do not need to say print you can simply, in this case, you can say s1 dot let us say, grade, it will give you the grade.

If I, if I want to access anything else which we have not created, it will not be available. So, for example, if I say s1 dot let us say gender, it tells me that s1 the student object has no attribute called gender, ok? So, in order to access this you need to create these, these attributes, ok?

(Refer Slide Time: 11:18)



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

```
[17]: s1.gendre

-----
AttributeError                                Traceback (most recent call last)
<ipython-input-17-2a29f899c43a> in <module>
----> 1 s1.gendre

AttributeError: 'Student' object has no attribute 'gendre'

[18]: ## Attributes of s2
s2.name = 'Sara'
s2.roll = 'MAT111'
s2.DOB = 'Aug 26, 1999'
s2.average = 85.6

[19]: print(s2.roll)
print(s2.average)

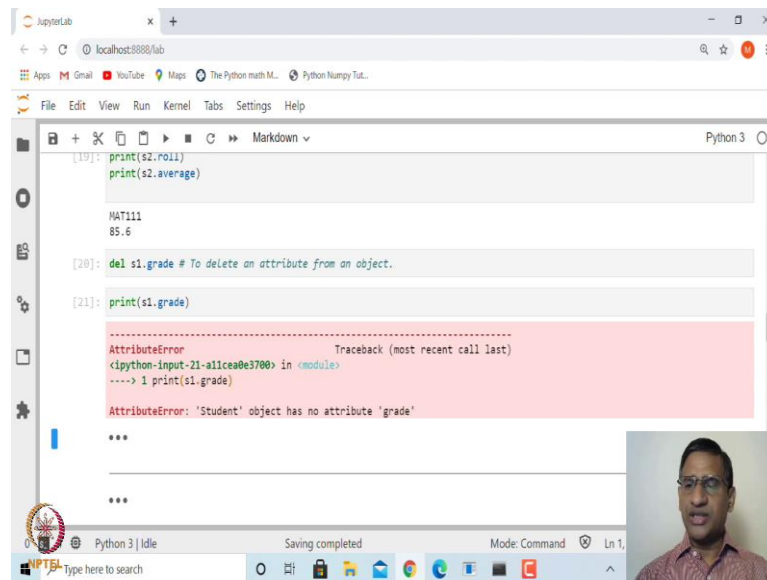
MAT111
85.6
***
```

The output of the code shows the attributes of s2: MAT111 and 85.6. A small video inset of a man is visible in the bottom right corner of the JupyterLab window.

Similarly, you can create another student let us say s, we have already created another student s2, right, with the same type student, and in that we can add again some attribute, and it is not necessary that s1 and s2 have the same attributes, you can have different attributes of the same, coming of the same class you can have different attributes, object can have different attributes, right?

So, for example, in this case, the attribute is name, which is Sara, roll which is MAT 111, which is, and then date of birth is August 26, 1999, and it has another attribute called average, which is let us say 85.6, ok? So, again you can print these, these things, these attributes from s1 of s1 using dot operator, right?

(Refer Slide Time: 12:36)



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

```
[19]: print(s2.roll)
      print(s2.average)

MAT111
85.6

[20]: del s1.grade # To delete an attribute from an object.

[21]: print(s1.grade)

-----
AttributeError                                Traceback (most recent call last)
<ipython-input-21-a1lcea0e3700> in <module>
----> 1 print(s1.grade)

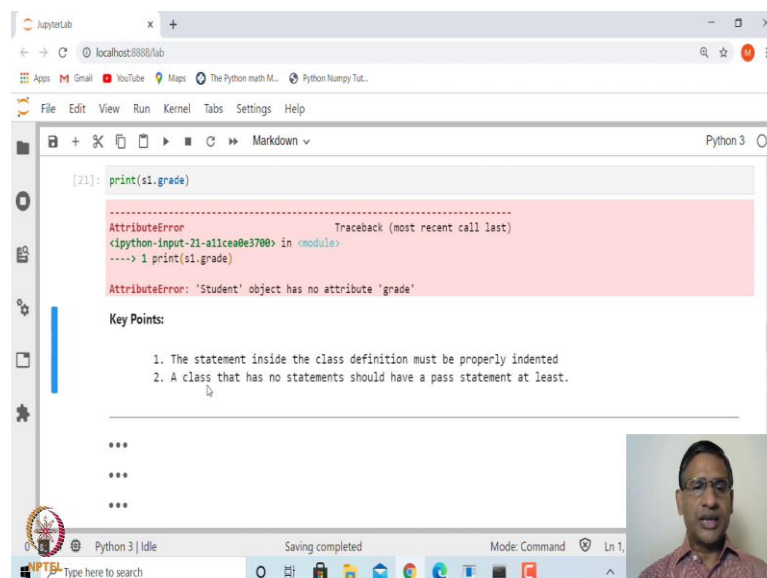
AttributeError: 'Student' object has no attribute 'grade'
```

The output shows the result of the first two cells, followed by the error message for the third cell. A small video inset of a man is visible in the bottom right corner.

And in this case, suppose you, at some point if you want to delete some attributes. So, for example, let us say we want to delete grade of the first student s1. So, that you can achieve using del function, del space s1 dot grade. Now, if you ask for what is this grade of s1, it will tell you that this particular attribute does not exist, the student object has no attribute grade.

Now, you can see that this we have not created this del for this student class, but it is available. So, there are some default attributes we will look at several of them, ok, right?

(Refer Slide Time: 13:26)



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

```
[21]: print(s1.grade)

-----
AttributeError                                Traceback (most recent call last)
<ipython-input-21-a1lcea0e3700> in <module>
----> 1 print(s1.grade)

AttributeError: 'Student' object has no attribute 'grade'
```

Below the error message, there is a section titled "Key Points:" with two numbered items:

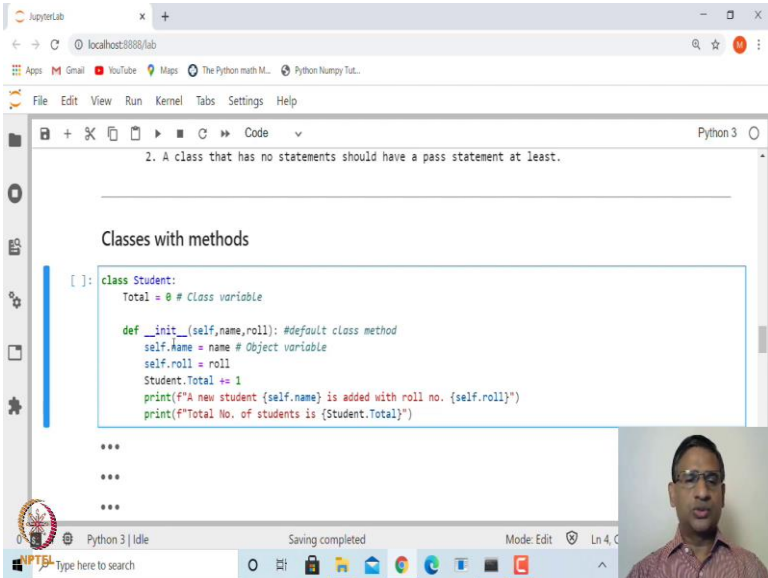
1. The statement inside the class definition must be properly indented
2. A class that has no statements should have a pass statement at least.

The output shows the result of the first cell, followed by the error message for the second cell. A small video inset of a man is visible in the bottom right corner.

So, if you look at the, how we have defined this student class. So, there are two things that you have to remember; first, you have to put your colon, and then all these statements should be indented inside, ok, right. So, that is these are the things you, you must keep in mind.

Now, let us look at how we can create a non-empty class, our class in which we want to define some methods, ok? For example, in case of complex class, you saw there is a method called conjugate. Now, in this case, student class let us say we want to define some, some methods. So, how do we do that?

(Refer Slide Time: 14:13)



```
2. A class that has no statements should have a pass statement at least.

Classes with methods

[ ]: class Student:
    Total = 0 # Class variable

    def __init__(self, name, roll): #default class method
        self.name = name # Object variable
        self.roll = roll
        Student.Total += 1
        print(f"A new student {self.name} is added with roll no. {self.roll}")
        print(f"Total No. of students is {Student.Total}")

    ***
    ***
    ***
```

Let us see. So, what we can do is first again let us define class, and space student colon, and let me give you, let, let us give variable total is equal to 0. So, this is actually called class variable, and we are initializing it equal to 0, so what we are thinking is the total number of students, right? Now, we can, so the basic idea is, as you create the objects inside this class, it each time it should add the number of students, ok?

Now, next is you, you, you create method which is called init which is get a default method. This is also known as initialization, right? So, how do we create? You just type in def, and space two underscores, two underscores init followed by again two underscores.



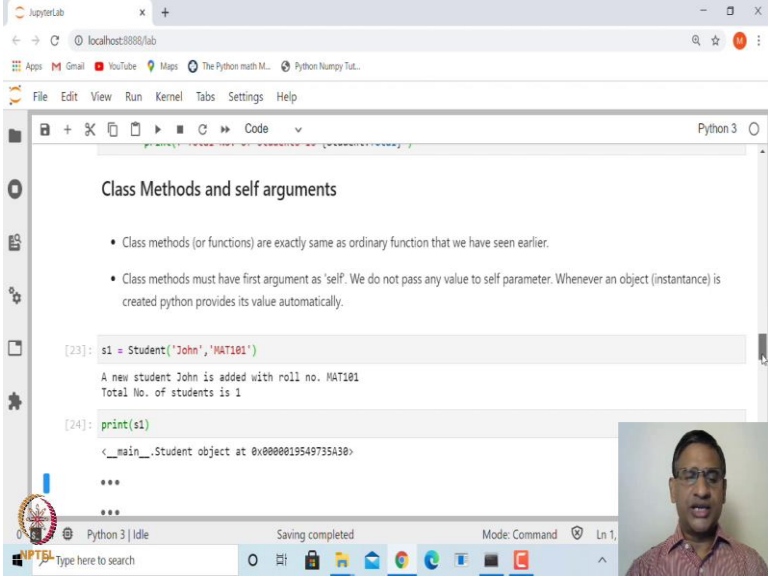
And then inside the bracket you have to give the arguments. By default this will have an argument called self. So, this is a default argument you have to give it, and then, we are giving two variables here which is the name, and roll, and these are the, these are called object variables.

So, how do we write this? So, we have to say self dot name is equal to the name that parameter which has been passed, and self dot roll is equal to roll that you have entered, ok? And then every time a student is added or an, an instance or object is created, you want to add the total number of student, right?

So, add this, and since this is, this is a class variable, this class variable, you have to access using student dot total, student dot total plus equal to 1; that means, the student will increment by 1, and then let us also print once a student is added, you print that a student with whatever name is added with whatever roll number, and then you also print what is the total number of students, ok?

So, let us create this. Let us create this, and once we have created this student class then we can keep on adding the different students that means we can create objects as many objects as you want.

(Refer Slide Time: 17:00)



```
Class Methods and self arguments
```

- Class methods (or functions) are exactly same as ordinary function that we have seen earlier.
- Class methods must have first argument as 'self'. We do not pass any value to self parameter. Whenever an object (instance) is created python provides its value automatically.

```
[23]: s1 = Student("John", "MAT101")
A new student John is added with roll no. MAT101
Total No. of students is 1

[24]: print(s1)
<__main__.Student object at 0x0000019549735A30>

***
```

And of course, in this case, as you saw this here this method init has one argument which is 'self', and we do not actually pass any value to this parameter, ok?

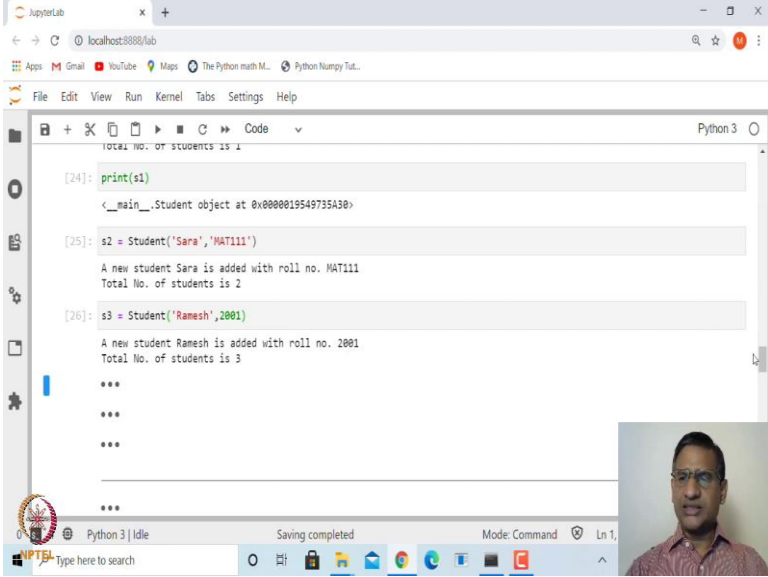
So, when we create an object we do not need to say s1 is equal to student, and then in the bracket self. We have to simply give the name, and the roll number, right? So, that is what it means. So, whenever an object is created, Python will provide its value automatically, that is, that is the role of self dot name, self dot roll, ok?

So, let us create an object, then object is s1, which is student, the name is John, and its roll number is MAT 101. So, let us run this. Now you can see here, the moment I have added one student with name John with roll number MAT 101, it gives me this print; a new student John is added with roll number this, and the total number of student is 1, that is what we have done. We have added only one student, and that is coming from these two print statements, right?

Suppose we create another student. So, now, we can also print what is the s1. Now, when I say print s1 it does not give me anything, right? Whereas, in case of z, when you created z, and when you asked for print z, it actually printed.

So, we will come to this when we have created this particular class, we do not know what is meaning of print, and that can be taken care using some other method, that we will come, come to it later, ok?

(Refer Slide Time: 19:03)



```
[24]: print(s1)
<__main__.Student object at 0x0000019549735A30>

[25]: s2 = Student('Sara', 'MAT111')
A new student Sara is added with roll no. MAT111
Total No. of students is 2

[26]: s3 = Student('Ramesh', 2001)
A new student Ramesh is added with roll no. 2001
Total No. of students is 3

***

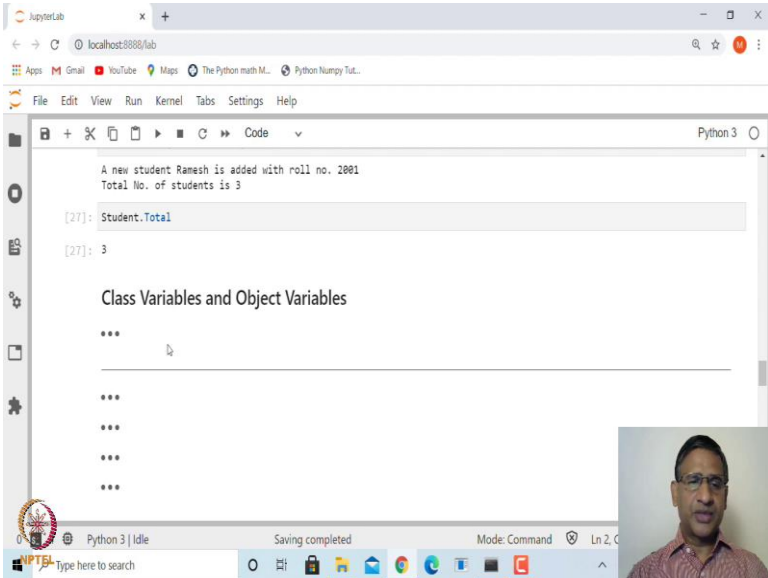
***

***
```

Now, let us say, create another student called Sara, and with roll number MAT 111. So, when we create this, what will it display? It will display that a new student called Sara is added, and with roll number this, and now total number of students is 2, ok?

So, if you create one more student, it will say let us say another student s3 which is equal to student, student, and let us say its name is Ramesh, and its roll number is let us say 2001, ok? So, now it said that Ramesh is added with roll number 2001, and number of student is 3, ok? So, that is the way one can create class with initialization, and then one can keep on adding the attributes to this class, ok?

(Refer Slide Time: 20:07)



The screenshot shows a JupyterLab window with a browser address bar at localhost:8888/lab. The interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar. The main code editor displays the following Python code:

```
A new student Ramesh is added with roll no. 2001
Total No. of students is 3

[27]: Student.Total

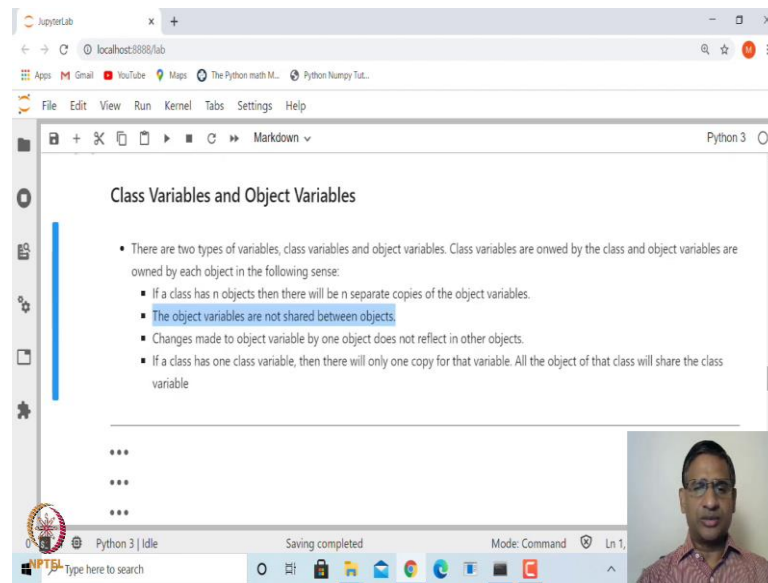
[27]: 3
```

Below the code, the text "Class Variables and Object Variables" is visible, followed by several lines of asterisks (\*\*\*). The bottom status bar indicates "Python 3 | Idle", "Saving completed", and "Mode: Command". A small video inset of a man is visible in the bottom right corner.

You can access total number of student which is a class variable in this case, using student dot total. So, if I say student dot total, now the 3 students have been added, so it is, gives, it gives you total, ok? And so, in this case, as you can see, we have created two types of variables, one is class variable, which is outside any method, and then you have object variables.

For example, here in this case, name, and roll number, they are all object variables, and, or the attributes of this particular class this particular student class, so for each student you can have different, different attributes, for each objects you can have different attributes. Whereas, this class variable, that will be common to all the, the objects. That is the difference between class variable, and object variable, ok?

(Refer Slide Time: 21:10)



So, that is what is explained here. There are two types of variables class variables, and object variables, class variables are owned by the class, and object variables are owned by each object. If the class has n number of objects, for example, in this case, we created 3 objects, then there will be n separate copies of object variables, and the object variables are not shared between objects.

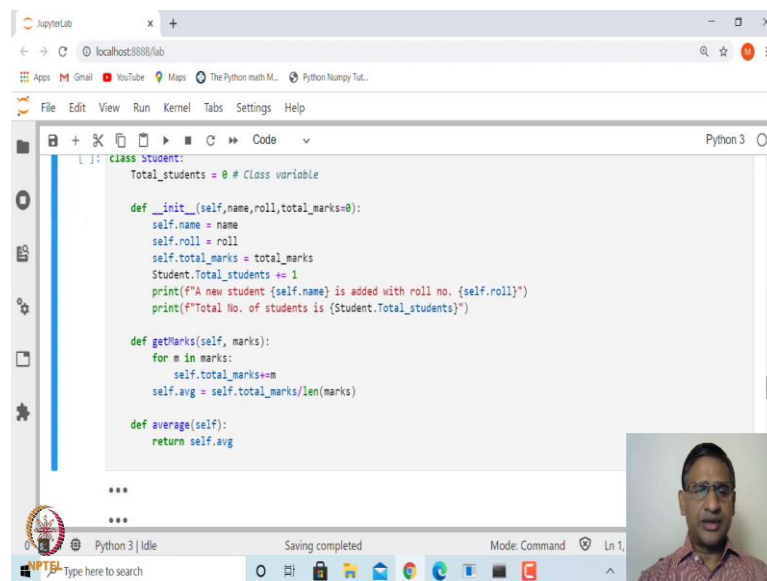
So, for example, we saw that, that s2, and s1 have different attributes. So, if for example, earlier we created s2, attributes for s2 as well average, whereas one attribute for s1 was grade.

But if you ask for s1 dot average it will not give you. Similarly, if you say s2 dot grade it will not give you anything; it will say attributes are not available, ok? So, the object variables are not shared between the different objects, whereas, class variables are common to all the objects.

Now, and changes made to an object, just one second, changes made to object variable by one object does not reflect in other objects; you can change, for example, we have created two variables s1, and s2, if I change their roll number of s2, the roll number of s1 will remain unchanged, right?

If a class has one class variable, then there will be only one copy of variable, and all objects of that class will share the class variable, right? That is what we saw; we have

here one class variable called total, and that is common to all the, the, the objects, ok?  
(Refer Slide Time: 22:59)



```
class Student:
    Total_students = 0 # Class variable

    def __init__(self, name, roll, total_marks=0):
        self.name = name
        self.roll = roll
        self.total_marks = total_marks
        Student.Total_students += 1
        print(f"A new student {self.name} is added with roll no. {self.roll}")
        print(f"Total No. of students is {Student.Total_students}")

    def getMarks(self, marks):
        for m in marks:
            self.total_marks += m
        self.avg = self.total_marks / len(marks)

    def average(self):
        return self.avg
```

Next, let us create another method inside this student class. We, we created one default method called initialization, but let us create our own method, and let us say, we want to create a method called getMarks, to get marks of the of a particular student, and this marks we will get as a list of marks; it may be list of marks in 5 subjects or 4 subjects or as many subjects as you want.

And then another one, we want to, once you have given the marks of the student, then we want to print the average of the marks. So, these are the two new methods we are creating inside this student class. So, how do we create? So, again creating method is very similar to creating a function. So, we will say def the name as getmarks.

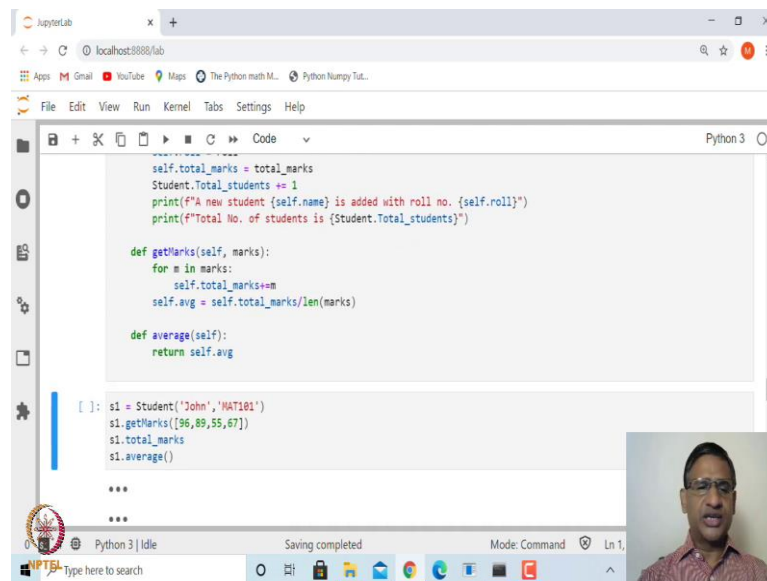
And again, the first parameter will be self, this is default parameter, and then this list of marks I am calling it as variable marks, and then now, so, let us create a variable called total marks, total marks, and how do we create that total marks? So, in, in this case, already we have created one variable in initialization called total marks, and which we initialized as 0 marks, initially it has 0 marks.

And then, so for m in marks that is the list, you simply say self dot total underscore marks plus equal to m; that means, for every m inside this mark, it will get added to this total marks, and then let us print the, the average, let us create the average, which is self dot average.

So, this is what? The total marks divided by the number of subjects for which you have entered the marks.

And then create another method called average, and that we are not passing any argument except that self, except that default argument self, and what should it return? It should return self dot average which we have calculated here, ok?

(Refer Slide Time: 25:27)



```
self.total_marks = total_marks
Student.Total_students += 1
print(f"A new student {self.name} is added with roll no. {self.roll}")
print(f"Total No. of students is {Student.Total_students}")

def getMarks(self, marks):
    for m in marks:
        self.total_marks += m
    self.avg = self.total_marks / len(marks)

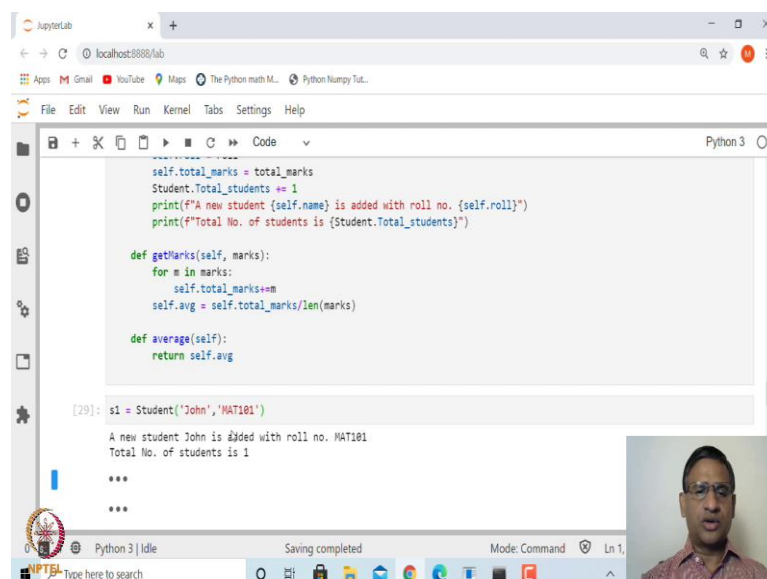
def average(self):
    return self.avg

[ ]: s1 = Student('John', 'MAT101')
s1.getMarks([96, 89, 55, 67])
s1.total_marks
s1.average()

***
***
```

So, let us run this. So, when we run this, now let us create an object called s1. So, let me go one by one.

(Refer Slide Time: 25:35)



```
self.total_marks = total_marks
Student.Total_students += 1
print(f"A new student {self.name} is added with roll no. {self.roll}")
print(f"Total No. of students is {Student.Total_students}")

def getMarks(self, marks):
    for m in marks:
        self.total_marks += m
    self.avg = self.total_marks / len(marks)

def average(self):
    return self.avg

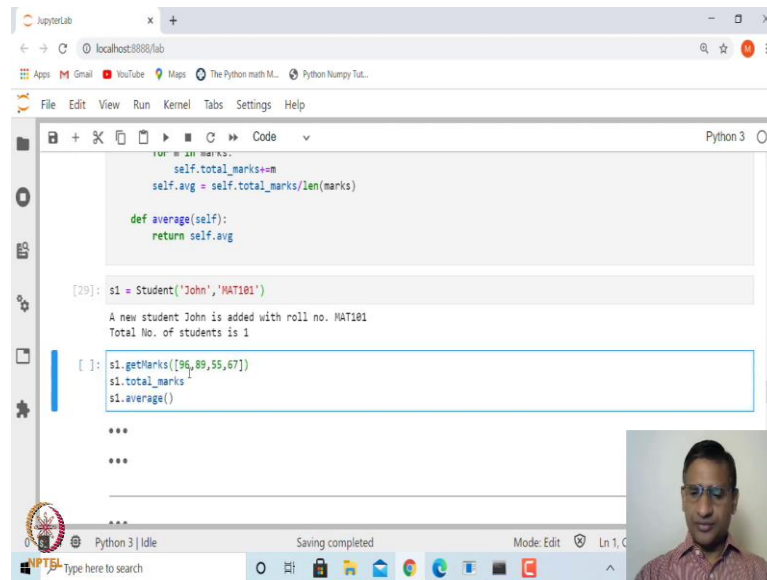
[29]: s1 = Student('John', 'MAT101')

A new student John is added with roll no. MAT101
Total No. of students is 1

***
***
```

So, this is an object called `s1` which is student with named John, and the roll number MAT 101, and again it is displaying these two messages that is, that is coming from these two print statement which we have created already inside this initialization itself, right, and after that let us get the marks.

(Refer Slide Time: 26:00)



```
class Student:
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no
        self.total_marks = 0
        self.avg = 0
        self.marks = []

    def add_marks(self, marks):
        self.marks.extend(marks)
        self.total_marks += sum(marks)
        self.avg = self.total_marks / len(marks)

    def average(self):
        return self.avg

s1 = Student('John', 'MAT101')

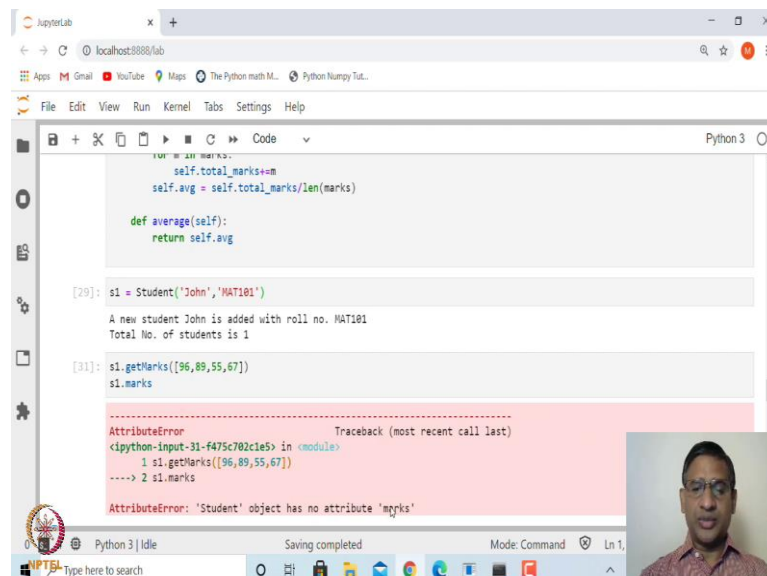
A new student John is added with roll no. MAT101
Total No. of students is 1

[ ]: s1.getmarks([96,89,55,67])
s1.total_marks
s1.average()

***
***
```

So, how do we get the marks? We have to say, we have to say `s1 dot getmarks`, and what is the parameter we need to pass? We need to pass parameter marks as a list, that is what we, we have decided.

(Refer Slide Time: 26:15)



```
class Student:
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no
        self.total_marks = 0
        self.avg = 0
        self.marks = []

    def add_marks(self, marks):
        self.marks.extend(marks)
        self.total_marks += sum(marks)
        self.avg = self.total_marks / len(marks)

    def average(self):
        return self.avg

s1 = Student('John', 'MAT101')

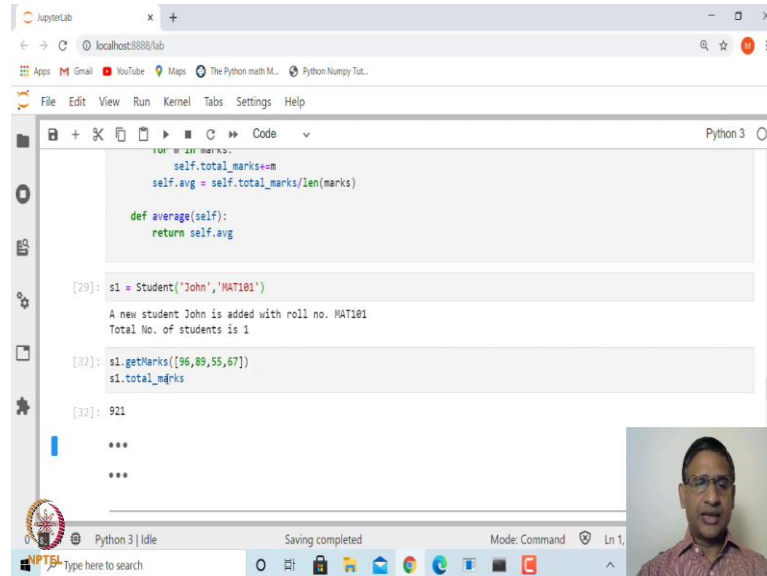
A new student John is added with roll no. MAT101
Total No. of students is 1

[31]: s1.getmarks([96,89,55,67])
s1.marks

AttributeError: 'Student' object has no attribute 'marks'
```

So, this is the, now if I say s1 dot marks, it does not have any, any attribute marks that is why you are not able to access, ok?

(Refer Slide Time: 26:28)



```
class Student:
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no
        self.total_marks = 0
        self.avg = 0

    def getMarks(self, marks):
        self.total_marks += sum(marks)
        self.avg = self.total_marks / len(marks)

    def average(self):
        return self.avg

[29]: s1 = Student('John', 'MAT101')
A new student John is added with roll no. MAT101
Total No. of students is 1

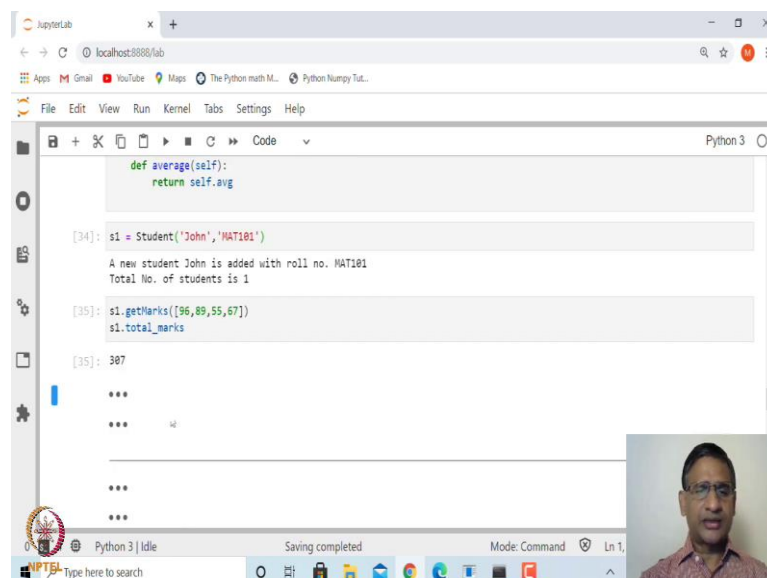
[32]: s1.getMarks([96,89,55,67])
s1.total_marks

[32]: 921

***
***
```

Now, let us look at what is s1 dot total marks, total marks. This we have defined, and this is total marks is 921. I think there is something wrong in this. Let me run this again, and let us call this s1, and get marks, that is correct.

(Refer Slide Time: 26:58)



```
def average(self):
    return self.avg

[34]: s1 = Student('John', 'MAT101')
A new student John is added with roll no. MAT101
Total No. of students is 1

[35]: s1.getMarks([96,89,55,67])
s1.total_marks

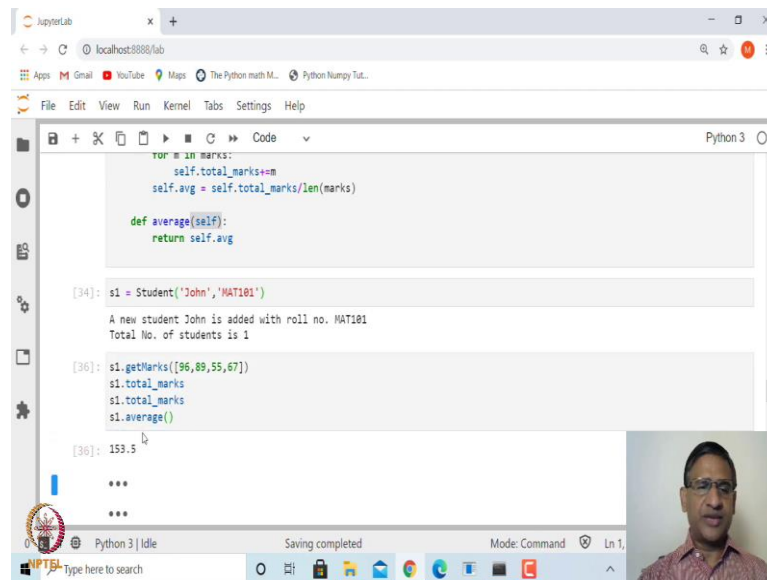
[35]: 307

***
***
***
```

So, it was actually initialized as something, right. So, the total marks is 307.



(Refer Slide Time: 27:07)



```
class Student:
    total_marks = 0
    avg = 0

    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no
        self.total_marks = 0
        self.avg = 0

    def getMarks(self, marks):
        self.total_marks += sum(marks)
        self.avg = self.total_marks / len(marks)

    def average(self):
        return self.avg

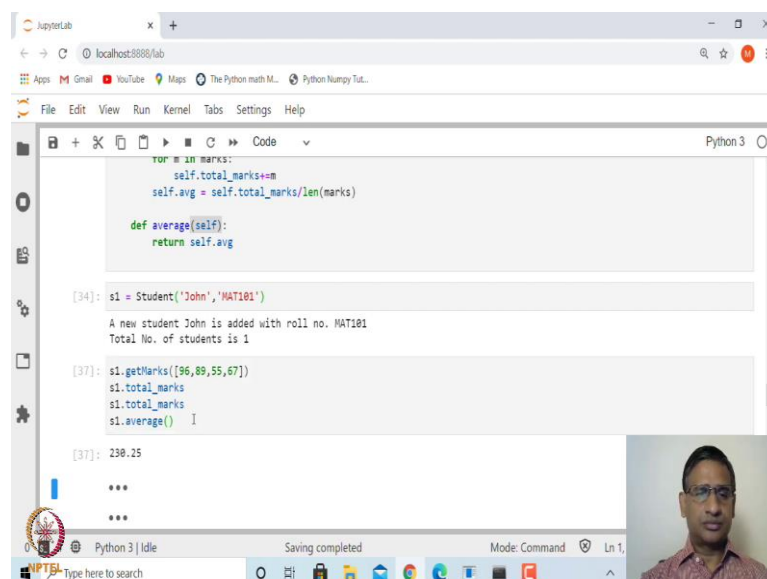
s1 = Student('John', 'MAT101')
A new student John is added with roll no. MAT101
Total No. of students is 1

s1.getMarks([96,89,55,67])
s1.total_marks
s1.total_marks
s1.average()

[36]: 153.5
```

If I want to get what is the average, I can simply now access that average method as `s1` dot `average`, and empty round bracket. Why only empty round bracket? Because we have not asked for any parameter here, ok, any argument. So, we will simply say `s1` dot `average`, this is the last one which is printing; `s1` dot `average` is, ok? So, you can see here, this is keeps on increasing because we are adding these marks. So, that is why it is, it is printing.

(Refer Slide Time: 27:32)



```
class Student:
    total_marks = 0
    avg = 0

    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no
        self.total_marks = 0
        self.avg = 0

    def getMarks(self, marks):
        self.total_marks += sum(marks)
        self.avg = self.total_marks / len(marks)

    def average(self):
        return self.avg

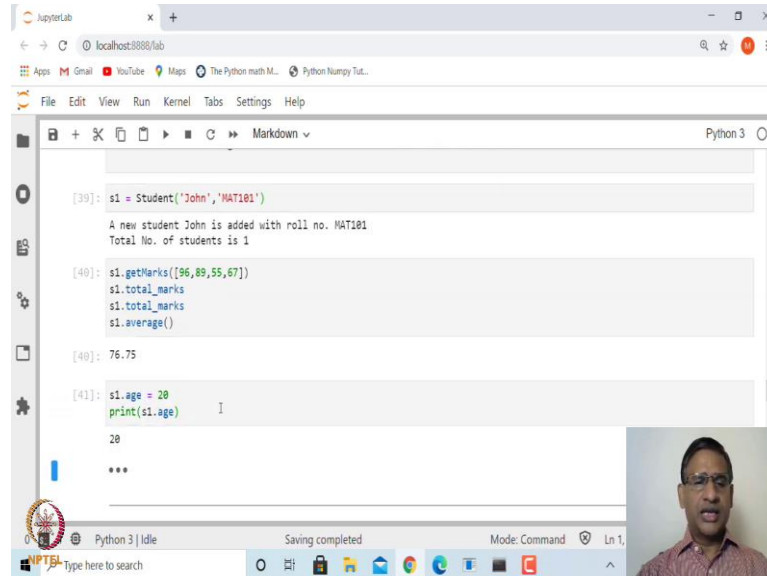
s1 = Student('John', 'MAT101')
A new student John is added with roll no. MAT101
Total No. of students is 1

s1.getMarks([96,89,55,67])
s1.total_marks
s1.total_marks
s1.average()

[37]: 238.25
```

So, let us reinitialize, reinitialize, and then when we say this average the average is 76.75. Similarly, you can add another student, and create its average.

(Refer Slide Time: 27:46)



```
[39]: s1 = Student('John', 'MAT101')
A new student John is added with roll no. MAT101
Total No. of students is 1

[40]: s1.getMarks([96, 89, 55, 67])
s1.total_marks
s1.total_marks
s1.average()

[40]: 76.75

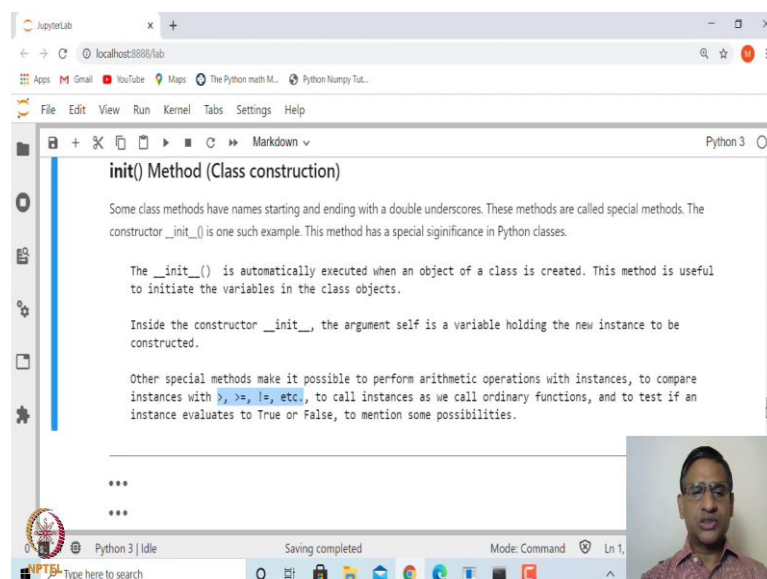
[41]: s1.age = 20
print(s1.age)

20
***
```

Now, if I asked for s1 dot age is equal to 20, though we have not created the age variable here, age is not a variable, ok? It is not an attribute.

But suppose if I say s1 dot age is equal to 20, then it will add, ok? So, even though you have initialized using init method still you can add attributes to any, any object, that is what it means, right?

(Refer Slide Time: 28:32)



### init() Method (Class construction)

Some class methods have names starting and ending with a double underscores. These methods are called special methods. The constructor `__init__()` is one such example. This method has a special significance in Python classes.

The `__init__()` is automatically executed when an object of a class is created. This method is useful to initiate the variables in the class objects.

Inside the constructor `__init__`, the argument `self` is a variable holding the new instance to be constructed.

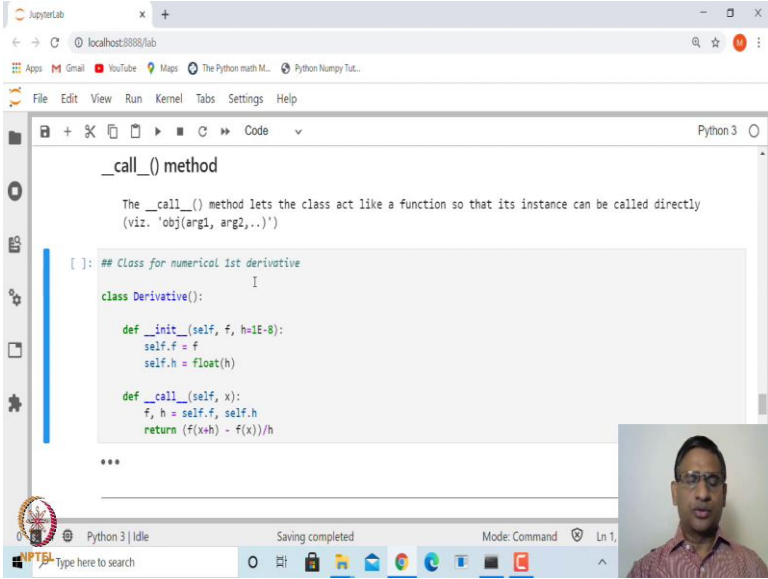
Other special methods make it possible to perform arithmetic operations with instances, to compare instances with `>`, `>=`, `!=`, etc., to call instances as we call ordinary functions, and to test if an instance evaluates to True or False, to mention some possibilities.

```
...
...
```

So, if you look at this init method, this is what is called a class construction, this is a constructor method, and so, it, it always starts with two underscores with name init, and then again two underscores, and inside the bracket you write the arguments, first self, and then followed by the arguments or initial, initial values, right? So, this is what is called special method. This is known as special method.

And as soon as an object is created this method will get executed, ok? And this method is useful to initiate, initialize the variables of the class that is what I said, ok? Inside the, this init constructor method, the argument self is a variable, holding the new instance to be constructed, right, and there are several other special methods. Some of the methods are for comparing objects, some of the methods are for printing, and so on. So, we will, we will see a few of them.

(Refer Slide Time: 29:51)



```
__call__() method

The __call__() method lets the class act like a function so that its instance can be called directly
(viz. 'obj(arg1, arg2,...)')

[ ]: ## Class for numerical 1st derivative
      I
class Derivative():

    def __init__(self, f, h=1E-8):
        self.f = f
        self.h = float(h)

    def __call__(self, x):
        f, h = self.f, self.h
        return (f(x+h) - f(x))/h

...

Python 3 | Idle
Saving completed
Mode: Command Ln 1,
```

First, let us look at another method another special method called call, right? So, for example, when you created a function in, in a code Python, when you created a function, you could also call it, right?

Suppose if I create a function  $f(x)$  equal to some  $\sin x$  plus  $x$  square, then if I say  $f$  of 2, it will give me the value of the function. It will evaluate the value of the function at 2. So, this is what is inside this class, there is a method, a special method called call, and the role of this call is exactly that, ok? So, let us see how do we, how do we create that.

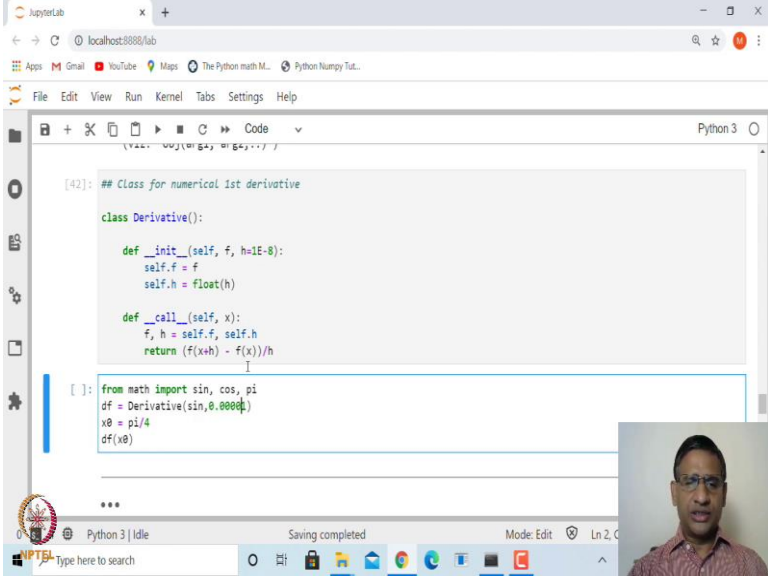
So, let us create a class to calculate numerical, first-order derivative, right? So, you must have seen the derivative as a limit of the difference quotient, that is, if you want to find derivative of a function  $f$  at a point  $x$ , then you look at  $f$  of  $x$  plus  $h$  minus  $f$  of  $x$  divided by  $h$ , and take the limit of this function, limit of this function as  $h$  goes to 0.

Therefore, as  $h$  becomes small and small, this  $f$  of  $x$  plus  $h$  minus  $f$  of  $x$  by  $h$ , this is approximately very close to the derivative of this function. So, that is what it returns.

And so, how do we create this, this class? So, class name is derivative, class name is derivative, and inside that you create the initialize, initial init method with `self`, and the function, and then this is the error term, that is the  $h$  value. So, by default, it is taking 10 to the power minus 8, you can change this, and then store this  $f$  in `self.f`, and  $h$  in `self.h`, and this somebody may give this integer value, so in that case, you convert into float, ok?

And then call these methods called `def`, again two underscores call, two underscores, and then `self`, and you want to call this at some point  $x$ . So, and what is it going to do? So,  $f$ , and  $h$ , I am just reinitializing,  $f$  and  $h$ , as `self.f`, `self.h`, and then return this, the difference quotient, ok?

(Refer Slide Time: 32:14)



```
[42]: ## Class for numerical 1st derivative

class Derivative():

    def __init__(self, f, h=1E-8):
        self.f = f
        self.h = float(h)

    def __call__(self, x):
        f, h = self.f, self.h
        return (f(x+h) - f(x))/h

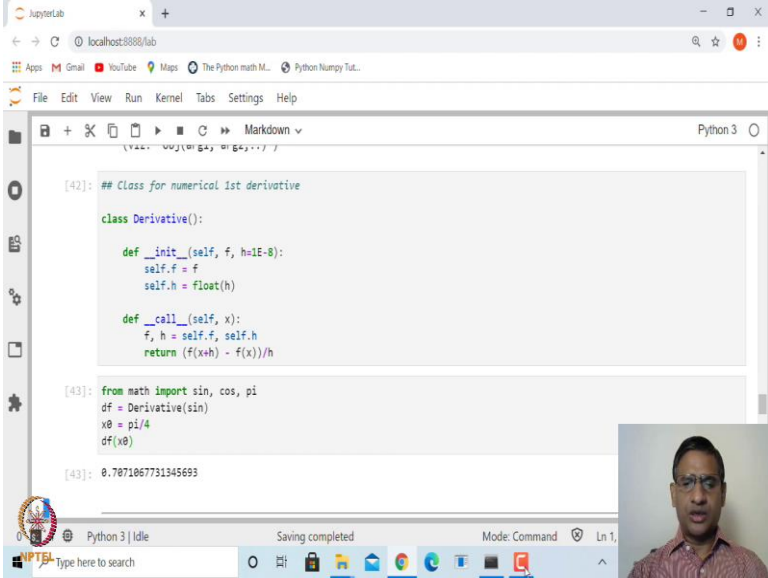
[ ]: from math import sin, cos, pi
df = Derivative(sin, 0.00001)
x0 = pi/4
df(x0)

***
```

So, let us run this. When you run this, now how do I call this derivative? So, let me, let us say, we want to find derivative of  $\sin x$ , ok? So, let us import sine and cosine, and we want to evaluate the derivative at, let us say,  $\pi$  by 4.

So, let us import this from math module, and then define  $df$  is equal to derivative of the function  $\sin$ , that is  $f$ , and this is the, the value of  $h$ . By default, it is not necessary to give this value, then in that case it will give, it will take the default value is 10 to the power minus 8.

(Refer Slide Time: 32:50)



```
[42]: ## Class for numerical 1st derivative

class Derivative():

    def __init__(self, f, h=1E-8):
        self.f = f
        self.h = float(h)

    def __call__(self, x):
        f, h = self.f, self.h
        return (f(x+h) - f(x))/h

[43]: from math import sin, cos, pi
df = Derivative(sin)
x0 = pi/4
df(x0)

[43]: 0.7071067731345693
```

The screenshot shows a JupyterLab window with a browser address bar at localhost:8888/lab. The code editor contains two cells. Cell [42] defines a class Derivative with an \_\_init\_\_ method that takes a function f and a step size h (default 1E-8), and a \_\_call\_\_ method that calculates the numerical derivative using the formula (f(x+h) - f(x))/h. Cell [43] imports sin, cos, and pi from the math module, creates an instance df of the Derivative class with sin as the function, sets x0 to pi/4, and calls df(x0). The output of the second cell is 0.7071067731345693. A small video inset of a man is visible in the bottom right corner of the JupyterLab window.

And so, the derivative has been created of  $\sin$ , and if I am to evaluate this at  $x_0$  is equal to  $\pi$  by 4, and in that case, we will say  $df(x_0)$ . So, that is what, that is how you call a function to evaluate at some point.