Scientific Computing Using Python

Professor. Vivek Aggarwal and Professor. Mani Mehra

Department of Mathematics

Indian Institute of Technology, Delhi

Lecture No. 01

Welcome to Scientific Computing Using Python. This course is called Scientific Computing Using Python. We will discuss different types of numerical methods used in computers with the help of Python programming.

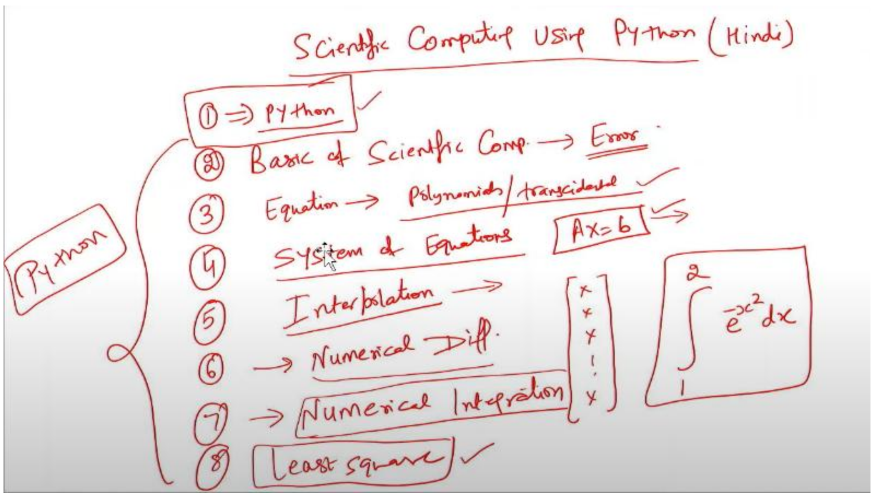
The outline of this course will be like this: Scientific Computing Using Python —this course is made in Hindi. In the first 2–3 weeks, we will discuss Python programming. Python is very popular these days. It is very useful and easy to handle. We will start with the basics. In the first 2–3 weeks, we will discuss Python. We will start with Python, and then we will discuss the basics of scientific computing.

The basics of scientific computing mean we will discuss how errors are involved in computing. We will implement algorithms on the computer. There are some inbuilt errors. There are some computational errors. We will discuss those errors. Then we will discuss how to solve equations.

For example, we have quadratic, cubic, fourth-order, polynomial, and transcendental equations. We will discuss how to find the roots of these equations. It won't be easy to find the roots of these equations with pen and paper. For this, we will discuss various numerical methods. We will find the roots of polynomials and transcendental equations.

We will discuss different types of numerical methods and their errors. Then we will discuss the system of equations. A system of equations means that if we have a system of equations like this, you must have solved a system of equations where the matrix is 3×3 or 2×2. But if the matrix is 4×4, 10×10, or 50×50, we know we must solve it on the computer.

(Refer slide time: 10:23)



Scientific Computing Using Python (Hindi)

① ⇒ Python ✓

② Basic of Scientific Comp. → Error

③ Equation → Polynomials / transcidental ✓

④ System of Equations $\boxed{Ax=b}$

⑤ Interpolation →

⑥ → Numerical Diff.

⑦ → Numerical Integration

⑧ Least square ✓

Python

$\begin{bmatrix} x \\ x \\ x \\ \vdots \\ x \end{bmatrix}$

$\boxed{\int_1^2 e^{-x^2}dx}$

We will use those methods to find the solution to such types of equations. We will solve these equations with numerical methods. Then, we will discuss the term Interpolation. Interpolation is a scientific way to find missing data. If we have any missing data, how do we fill in that data? For that, we use interpolation. Interpolation is used a lot in data science. If we have a large dataset and we want to utilize that data effectively, interpolation becomes very useful.

We will discuss interpolation, and then we will discuss differentiation, which we call numerical differentiation. In numerical differentiation, we have data. You know that data is often given in the form of a column. Suppose we have some data points, and someone asks us to differentiate these data points. We know that differentiation is the derivative of a continuous function. We will discuss how to take the numerical derivative. It is used a lot nowadays whenever you do any data-related work. If you want to observe the rate of change, you have to take the derivative. This is called numerical differentiation.

The next part is numerical integration. In numerical integration, suppose we have to calculate the integral of e to the power of minus x square dx between 1 and 2. We know that we cannot solve this using pen and paper, so what will we do? We will take the help of a computer. How is the computer solving it? The computer applies specific methods. We will discuss these methods as part of numerical integration.

We will study the theory of approximation and how we can do approximations. We will discuss the least squares method. The least squares method is used for data fitting. If we have data and we want to fit different types of functions to it, we will learn how to do this using least squares fitting.

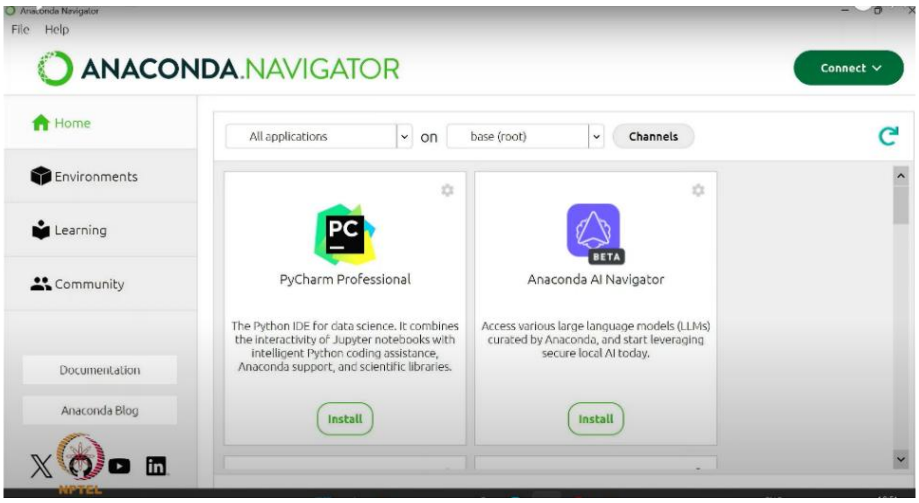
Along with all this, we will do Python programming. If we develop an algorithm to find the root of an equation, we will write the code in Python. We will compile it and discuss the output. After that, we will analyze the error. All of this will be done with the help of Python.

First, we will learn Python. What is Python? Python is a programming language. You may have used other programming languages, like C, Fortran, C++, Java, and MATLAB. These are all programming languages. Python is widely used because it is easy to understand and cost-free. For example, MATLAB is not free, although it is a very good language and easy to learn. Our second course, Scientific Computing using MATLAB, is based on MATLAB. But sometimes students face the problem of not having access to MATLAB. So why not use free software?

In Scientific Computing using Python, we decided to offer this course in Hindi for students. This is just an outline of our first course. We will start with Python. For Python, we need a compiler where we will write the code and compile it. We have downloaded Anaconda. We will do all this work in a Jupyter Notebook. You can also use different types of compilers, such as Google Colab.

Our first task is to start Python. To start Python, we will use the Anaconda Navigator.

(Refer slide time: 11:18)



The Python IDE for data science. It combines the interactivity of Jupyter notebooks with intelligent Python coding assistance, Anaconda support, and scientific libraries.

Access various large language models (LLMs) curated by Anaconda, and start leveraging secure local AI today.

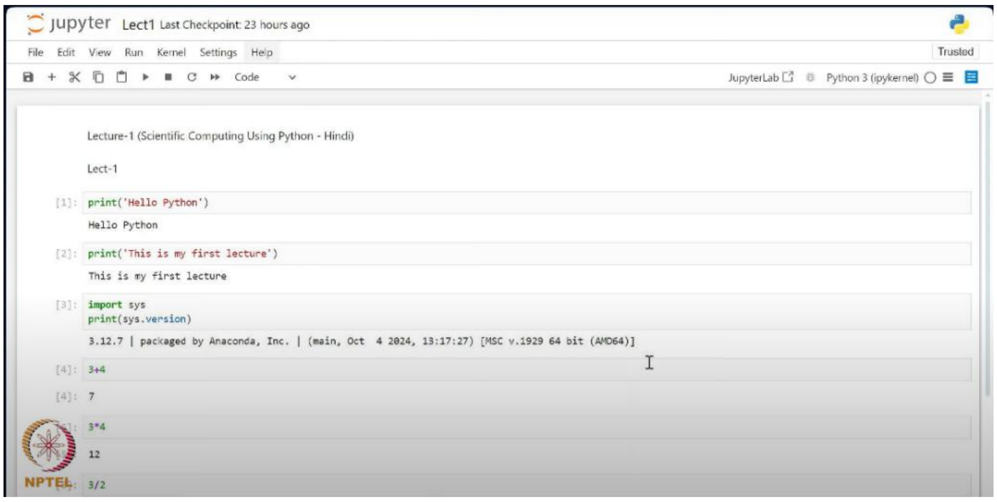
You can download it from the internet. You will get many videos on how to download it and get started. We have downloaded Anaconda. It is free. Download Anaconda and install it. After that, we will use Jupyter Notebook. We will click on launch, and it will launch. After launch, this is the compiler we have opened. It will look like this. Jupyter and lecture one, we have named it. We will write and compile our code in this space. In this sense, this software, written in Python, is very easy to understand. We can understand it very easily. So, as we have written in Lecture 1, we will learn basic commands in Python and how to use Python.

For Python, first, if I want to write, I will use Markdown. I can write that today's lecture is 1. If we want to compile, we will press Shift and Enter. Shift and Enter to compile it. Suppose we want to write something. I want to print. I will write Hello Python in inverted commas and press Shift and Enter. Hello Python print means whatever we write will be printed on the screen. We can write anything else. We can write in double commas or single commas. This is my first lecture, then press Shift and Enter to print, which is for printing.

As I downloaded Anaconda, we will see which version it is working in. It is written in Python 3. But if we want to see its version, I will write import sys, then I will write print(sys.version). We will see its version. It is written 3.1.2.7. It is written and packaged by Anaconda. All these things were downloaded in 64-bit on a computer. We got the data for our Python. In which version is it working? Your version might be old, but this is the updated version in which it is working, Python Anaconda 3.1.2.7.

We have seen its version. Now, we want to do some work. We want to do just addition. I wrote three plus 4, Shift and Enter, we got 7. I want to do three multiplied by 4, got 12. I divided 3 by 2, Shift and Enter, got 1.5. Like this, we can do all these mathematical operations easily. For three power 2, we have to use double star for power, and after entering, we get 9. 3 by 2 is 1.5, 3 multiplied by 4 is 12; all these are floating numbers.

(Refer slide time: 16:42)



Lecture-1 (Scientific Computing Using Python - Hindi)

Lect-1

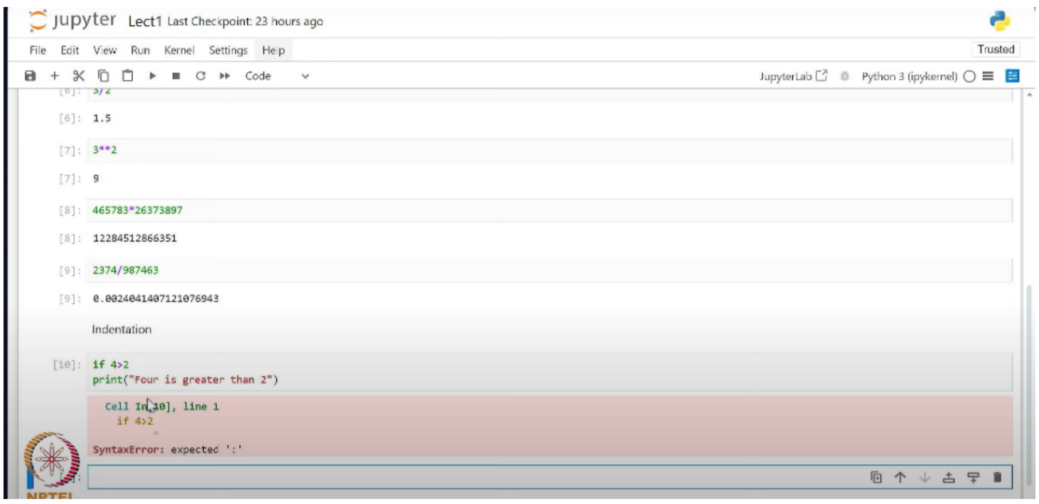
```
[1]: print('Hello Python')
     Hello Python

[2]: print('This is my first lecture')
     This is my first lecture

[3]: import sys
     print(sys.version)
     3.12.7 | packaged by Anaconda, Inc. | (main, Oct  4 2024, 13:17:27) [MSC v.1929 64 bit (AMD64)]

[4]: 3+4

[4]: 7

     3*4

     12

     3/2
```

We can do it very easily. There is nothing in it; whatever you can do on a calculator, you can do in it. You can do big calculations like 465783 by multiplying it by any number, and we get the result. You take a number and divide it; we get the result. We got the result. You can use it very easily like a calculator.

Slowly, we will see what type of commands it has. The first work we are going to do is Indentation. Indentation means how to do spacing. Like cell 1 2 3, these are the means by which we give input and get the output. Suppose I don't want to provide input, but write something. I will drop down here and get Markdown. In Markdown, it will be just a comment for our knowledge. We are using Indentation. We can write the indentation here, so I shift and enter here and get the indentation.

Indentation means how to do spacing like in if loop i write suppose 4 is greater than 2, so we know this is true after that i wrote in next command print, in print I write this " 4 is greater than 2 " I wrote this and I entered here so now see what we get (an error).

(Refer slide time: 20:00)



```
[6]: 1.5

[7]: 3**2

[7]: 9

[8]: 465783*26373897

[8]: 12284512866351

[9]: 2374/987463

[9]: 0.0024041407121076943
```

Indentation

```
[10]: if 4>2
      print("Four is greater than 2")

  Cell In[10], line 1
    if 4>2
          ^
SyntaxError: expected ':'
```

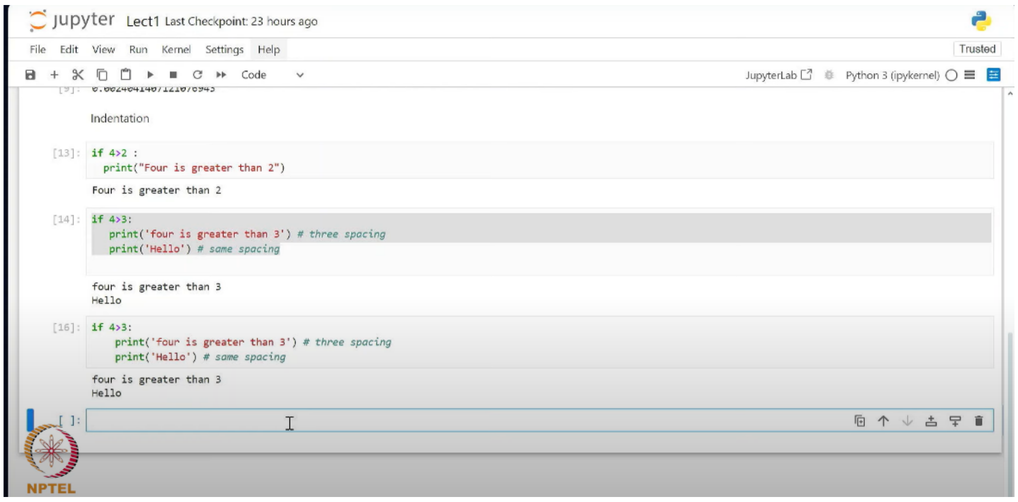
So what is the problem here in cell number 10, line 1, if 4 is greater than so we got unexpected expected this one ':', so what we have to do is we have to write a colon here. Hence, we wrote a colon after that. We ran now and saw what was written here. There is an indentation error expected an indented block after the if statement on line 1. This means line 1 is this, after this what we have indent indented block, so as you know indented block means if we started here if statement so following command in that block in that if loop we have to give spacing not on same level like I gave one space so one space means it is indented now I run this shift and enter see what is written 4 is greater than 2. So this is called an indentation error. It means if we are working on any function, like an if loop, in this, we don't have to write on the same level. Next statement we can give one spacing or two spacing like I gave one spacing i gave two spacing so i gave two spacing after that shift and enter so at least we have to keep one spacing in it we can keep 2, 3, 4 so all this but we can't keep it on same level so this is called indentation.

For example, if I wrote that 4 is greater than 3, I would write this and put a colon after that. I wrote print ('4 is greater than 3'), so we wrote this now. Suppose I want to remember how much spacing I gave. So, see how many spacing 1, 2, 3. 3 spacing so I can write like this 3 spacing using #. I wrote this hash, which means these are comments for our use; if we run it, it won't affect our code.

So, I wrote this now here I have to write print statement so I wrote print here I wrote hello I wrote this and here I kept same level same spacing so here same spacing and I ran it so here it is written 4 is greater than 3 and hello so it means both of them worked. So here, if the statement says that we took commands, we kept both commands at the same spacing, so if we keep it at the same spacing, it will work.

Now, suppose we were writing like this, I will copy it and paste it here, but I changed the spacing. Hence, if statement in that we wrote print command and print command after that I run it now what happened indentation error unexpected indent it means first statement we gave spacing of 3 but next we changed level and gave spacing so what will happen indentation error will come because if we are using any function or statement. We are giving commands, so they should be on the same indentation and at the same level. So, if I write it like this or do it on the same level as I did it on the same level, shift and enter so that it will run, and the output will be 4 is greater than 3. We can print it, and the indentation should be the same. By doing this, the output will be correct.

(Refer slide time: 26:02)


jupyter Lect1 Last Checkpoint: 23 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

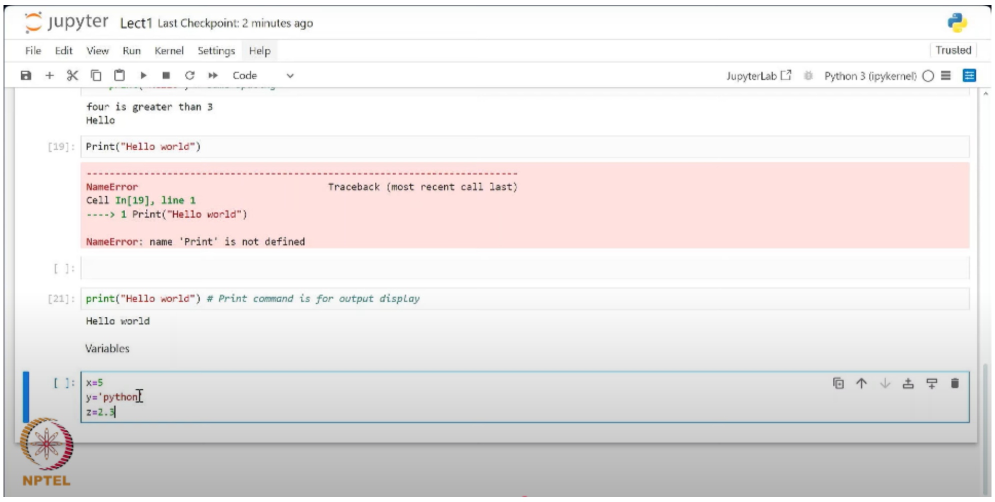
Indentation

```
[13]: if 4>2 :
        print("Four is greater than 2")

      Four is greater than 2

[14]: if 4>3:
         print('four is greater than 3') # three spacing
         print('Hello') # same spacing

      four is greater than 3
      Hello

[16]: if 4>3:
          print('four is greater than 3') # three spacing
          print('Hello') # same spacing

      four is greater than 3
      Hello

[ ]:
```

I wrote hash so hash is, suppose I am writing a command like print and in print I wrote "Hello world" but I don't remember why print command. So, what I do is I write here # print is to print the statement. So, for my understanding, this thing I wrote after hash means that I wrote it as a comment, so we call it a comment. So now we will run it. So, hello world is written, but this is for comment, it is not written. This means I want to tell myself that this print is just for comment by writing a hash, or I can write the print command for output display, so I will remember that the print command is used to display. Like this, we can write and print, and by writing hash, we can use comments for our use. This is just a basic command.

Now we will see how we can define a variable in Python. For this, we will mark down and here I define variables, which means I commented that we will use variables. Suppose I am writing x is equal to 5, so I defined the variable x and its value is 5. Similarly, I defined y and the given variable value 'python'. I defined variable z as equal to 2.3.

(Refer slide time: 29:20)


jupyter Lect1 Last Checkpoint: 2 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
      four is greater than 3
      Hello

[19]: Print("Hello world")

      ---------------------------------------------------------------------------
      NameError                                 Traceback (most recent call last)
      Cell In[19], line 1
      ----> 1 Print("Hello world")

      NameError: name 'Print' is not defined

[ ]:

[21]: print("Hello world") # Print command is for output display

      Hello world
```

Variables

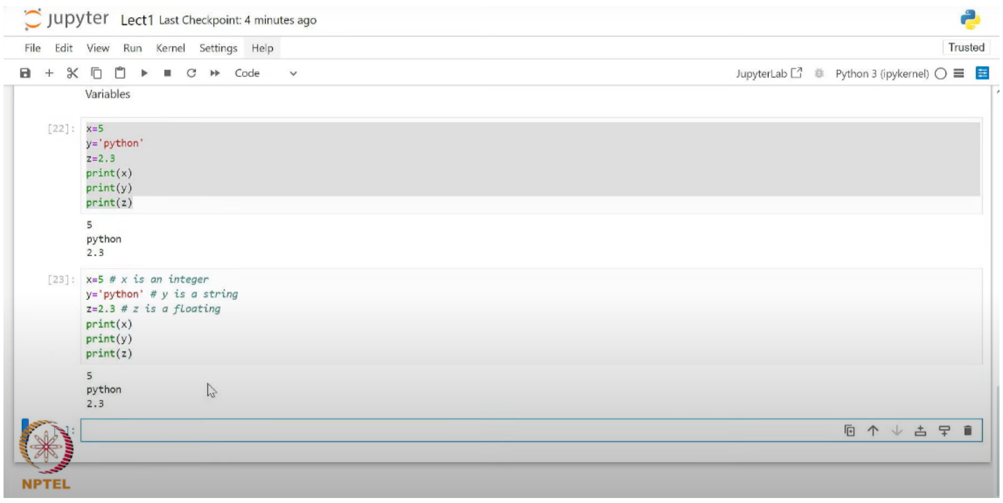
```
[ ]: x=5
     y='python
     z=2.3
```

Now I will print them. So, I printed x, printed y, and printed z, and then I entered it. After that, you can see the output: 5, Python, and 2.3. Now you know that 5 is in integer form, so x is an integer. The value of y is "python", which means it is a string, and this value is assigned to y.

The value of z is 2.3, which is a floating point. A floating point means it is in fractional form, so we call 2.3 a floating-point number. Hence, z is of float type.

To confirm this, I mentioned in comments that x is an integer, y is a string, and z is a floating point. Then I ran the program.

(Refer slide time: 31:09)


jupyter Lect1 Last Checkpoint: 4 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

Variables

```
[22]: x=5
      y='python'
      z=2.3
      print(x)
      print(y)
      print(z)

      5
      python
      2.3

[23]: x=5 # x is an integer
      y='python' # y is a string
      z=2.3 # z is a floating
      print(x)
      print(y)
      print(z)

      5
      python
      2.3
```

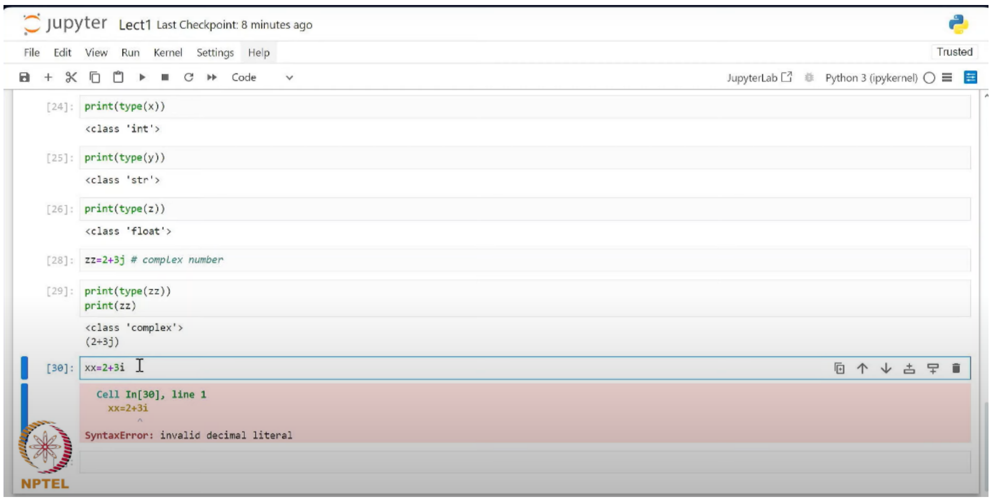
Now, if we want to see the type of each variable, I use the print function along with the type function. Print means it will display the output, and type means it will tell the data type of the variable. As soon as I checked the type of x, it showed "class int", which means x is an integer. Since Python is an object-oriented programming language, it uses classes to define types. So, "class int" means x is of integer type.

Then I checked the type of y, and it displayed "class str", which means y is a string. Similarly, when I checked the type of z, it displayed "class float", meaning z is a floating-point number.

Now I wanted to define another variable, so I took zz and assigned it the value 2 plus 3j. This means I am taking a complex number. The complex number has a real part and an imaginary part. Here, 2 is the real part and 3j is the imaginary part. In Python, we use 'j' to represent the imaginary part. So, 2 + 3j is a complex number. I also printed the value and the type, and it showed the value as 2 + 3j and the type as "class complex".

It is important to remember that in Python, we use 'j' and not 'i' to denote the imaginary part. If we write 2 + 3i, it will give a syntax error, saying "invalid decimal literal", because Python does not recognize 'i' for complex numbers.

(Refer slide time: 34:40)

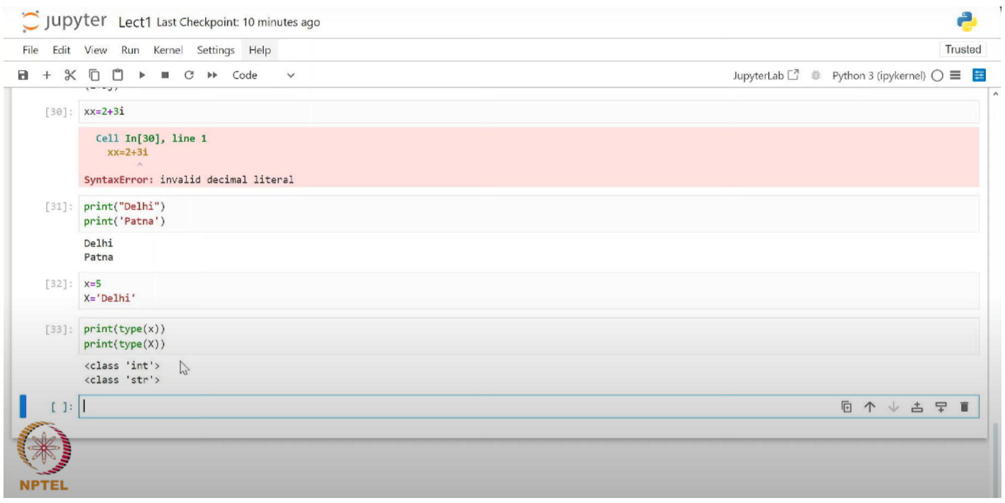

Jupyter Lect1 Last Checkpoint: 8 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[24]: print(type(x))
      <class 'int'>

[25]: print(type(y))
      <class 'str'>

[26]: print(type(z))
      <class 'float'>

[28]: zz=2+3j # complex number

[29]: print(type(zz))
      print(zz)
      <class 'complex'>
      (2+3j)

[30]: xx=2+3i
        Cell In[30], line 1
          xx=2+3i
            ^
      SyntaxError: invalid decimal literal
```

We can represent string using double inverted commas or single inverted comma. I wrote a print statement with double inverted commas and wrote "Delhi". Then I wrote another print statement with single inverted commas and wrote "Patna". When I ran them, both were printed correctly. This shows that whether we use single or double quotes, both are treated the same, and both represent string values.

Variable names in Python are case-sensitive. For example, I assigned 5 to small x and "Delhi" to capital X. When I checked their types, using print(type(x)) and print(type(X)) and enter. One showed integer and the other showed string. This means x and X are treated as two different variables.

(Refer slide time: 36:58)


Jupyter Lect1 Last Checkpoint: 10 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[30]: xx=2+3i
        Cell In[30], line 1
          xx=2+3i
            ^
      SyntaxError: invalid decimal literal

[31]: print("Delhi")
      print('Patna')
      Delhi
      Patna

[32]: x=5
      X='Delhi'

[33]: print(type(x))
      print(type(X))
      <class 'int'>
      <class 'str'>

[ ]:
```

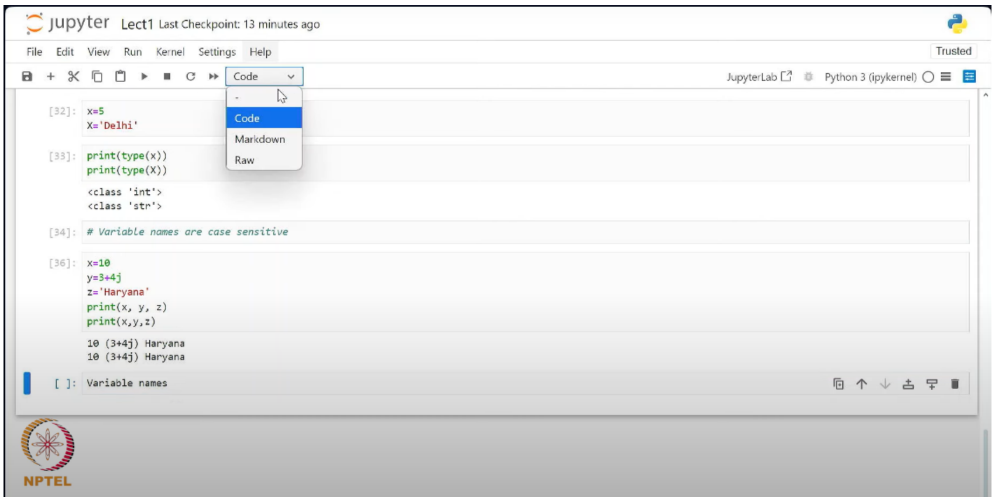
So, I wrote a comment stating that variable names are case-sensitive. Always keep this in mind and remember that variable names are case sensitive. When I run the code using Shift + Enter, it helps me understand and remember that variable names are case-sensitive.

Now I am writing different types of print, so I will see how to use the print command easily. As you have seen, I had to write it again and again—as I wrote above, print x, print y, print z—so I had to repeat it multiple times.

Suppose I have defined x as 10, y as 3 plus 4j, and z as ‘Haryana’. After that, I printed them by writing print(x, y, z) . So let us see what happens. It printed all of them at once, and there was spacing between them. The output was: 10, 3+4j, and Haryana. If I write it like this—print(x, y, z) its giving same output. So, whether we take spacing or not, it doesn’t matter; the output will come in this way.

So what happened here is that the print command printed all three values at once, and we didn’t have to write it again and again as we did before: print x, print y, print z. This saves effort. So, till now we get integer type, floating type, string type and complex type and got to now that these types of variable we use.

(Refer slide time: 40:21)

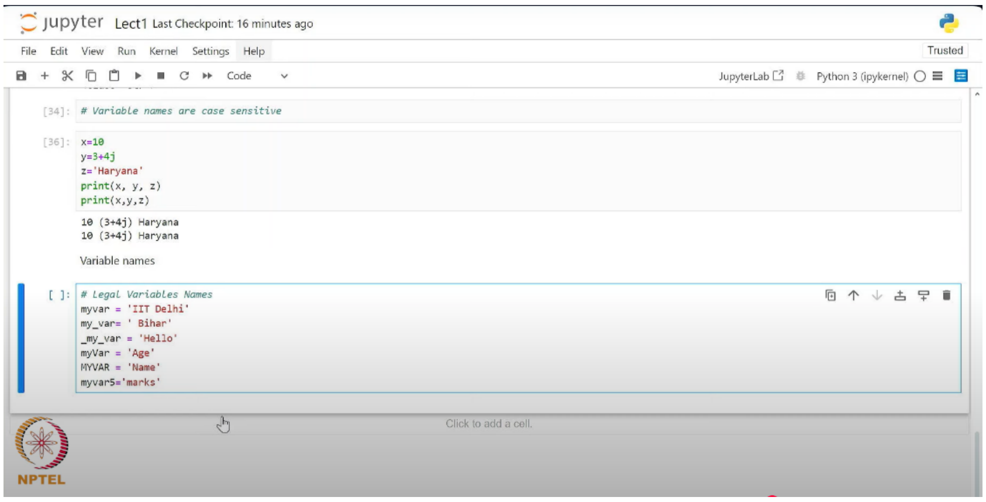

jupyter Lect1 Last Checkpoint: 13 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
-
Code
Markdown
Raw
JupyterLab
Python 3 (ipykernel)
NPTEL

```
[32]: x=5
      X='Delhi'

[33]: print(type(x))
      print(type(X))

      <class 'int'>
      <class 'str'>

[34]: # Variable names are case sensitive

[36]: x=10
      y=3+4j
      z='Haryana'
      print(x, y, z)
      print(x,y,z)

      10 (3+4j) Haryana
      10 (3+4j) Haryana

[ ]: Variable names
```

Now let us discuss variable names—how we can define variable names. It’s not that we can use just any name; we have to see which names are legal.

Let me first define legal variable names. Now, as I have written myvar = ‘IIT Delhi’, then I wrote my_var = ‘Bihar’, and the third one is _my_var = "Hello". Then I wrote another variable, say myVar = ‘Age’ (I had to define something, so I have define the variable age). I define with capital letter MYVAR = ‘Name’ and I define myvar5 = ‘marks’ . So we have define all of them like this: first we wrote myVariable together, then I used an underscore between my and variable, and in the third one, I started the variable name with an underscore.

(Refer slide time: 43:16)


jupyter Lect1 Last Checkpoint: 16 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Click to add a cell.
NPTEL

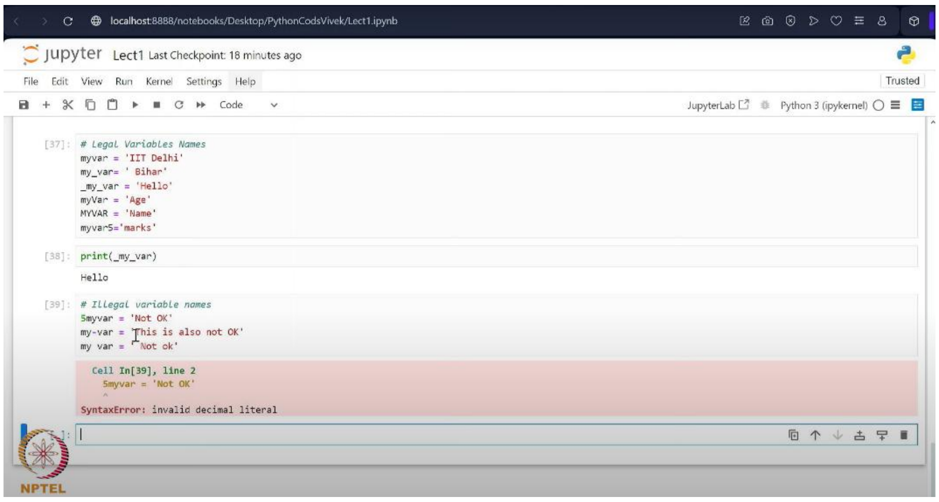
```
[34]: # Variable names are case sensitive

[36]: x=10
      y=3+4j
      z='Haryana'
      print(x, y, z)
      print(x,y,z)

      10 (3+4j) Haryana
      10 (3+4j) Haryana

      Variable names

[ ]: # Legal Variables Names
     myvar = 'IIT Delhi'
     my_var= ' Bihar'
     _my_var = 'Hello'
     myVar = 'Age'
     MYVAR = 'Name'
     myvar5='marks'
```

Now I ran it, and there was no error. So, if I print(_my_var), the output shows "hello", which means we can use all these names legally for our variables.

Now we will see what is illegal. In illegal names, the first thing is: we cannot start a variable name with a number. For example, 5myvar = ‘Not OK’ is not allowed. Similarly, if I write my-var = ‘This is also not OK because of the minus sign. Or if I leave a space and write my var = ‘Not ok’, that too is not valid.

Now, when I run this code, an error appears. It says that 5myVariable is not defined and it's an invalid decimal literal. So, this is not allowed. If I comment this out and run it, it shows my-var is also not allowed. After commenting my-var, if I run, we get spacing is also not allowed. So, all these things are not allowed if we are going to deal with variable names.

(Refer slide time: 45:33)



```
[37]: # Legal Variables Names
      myvar = 'IIT Delhi'
      my_var= ' Bihar'
      _my_var = 'Hello'
      myVar = 'Age'
      MYVAR = 'Name'
      myvar5='marks'

[38]: print(_my_var)

      Hello

[39]: # Illegal variable names
      5myvar = 'Not OK'
      my-var = 'This is also not OK'
      my var = ' Not ok'

        Cell In[39], line 2
          5myvar = 'Not OK'
          ^
      SyntaxError: invalid decimal literal
```

So, from this we learn that we cannot name variables using numbers at the beginning, use special characters like minus signs, or leave spaces in variable names.

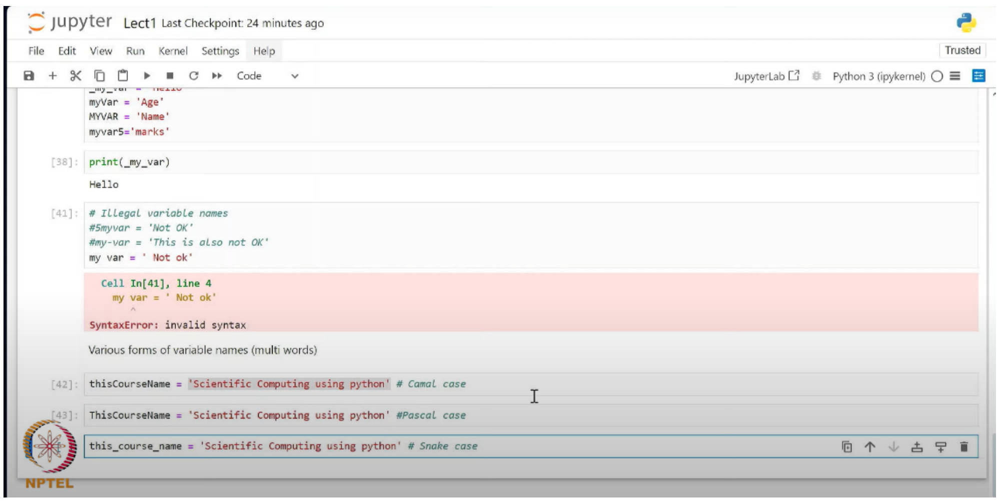
If we want to define a lot of variables, then how can we name them? Here, it is written as "Various forms of variable names".

There are multi-word variable names. So I will note this mark down and enter. Suppose I want to define a long variable name. I define this as thisCourseName, and assign it something like ‘Scientific Computing Using Python’. Now look at how I wrote this: the first word this is in small letters, then Course starts with a capital C, and Name starts with a capital N. This generate a string and its variable name is thisCourseName . This style is called Camel case so we comment Camel case. So, from the name thisCourseName, I know I’m talking about a course name.

Next, I write ThisCourseName, where each word starts with a capital letter. I copy this and define it again as ThisCourseName = ‘ Scientific Computing Using Python’. So now what I have done is capitalized the first letter of each word. This is called Pascal case.

In the third style, I write this_course_name = ‘Scientific Computing Using Python’, where I use underscores between the words. This is called Snake case. This is also allowed, and we use this style often in programming when we have to define many variables.

(Refer slide time: 51:04)


jupyter Lect1 Last Checkpoint: 24 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
NPTEL

```
myVar = 'Age'
MYVAR = 'Name'
myvar5='marks'

[38]: print(_my_var)

Hello

[41]: # Illegal variable names
#5myvar = 'Not OK'
#my-var = 'This is also not OK'
my var = ' Not ok'

  Cell In[41], line 4
    my var = ' Not ok'
      ^
SyntaxError: invalid syntax
```

Various forms of variable names (multi words)

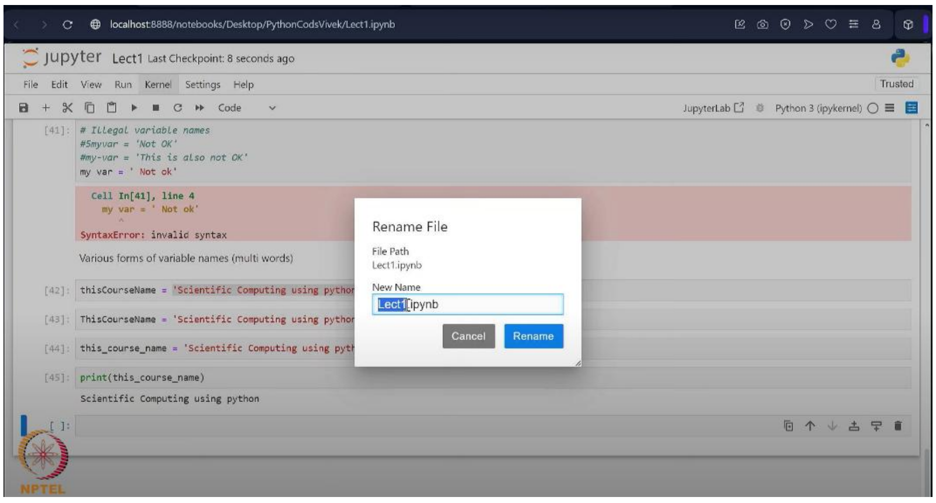
```
[42]: thisCourseName = 'Scientific Computing using python' # Camal case

[43]: ThisCourseName = 'Scientific Computing using python' #Pascal case

this_course_name = 'Scientific Computing using python' # Snake case
```

So, to define variables, we can use any of these forms—camelCase, PascalCase, and snake_case.

Now, if I print thisCourseName, the variable name, then the output will be displayed. Since this variable holds a string value, it will display the string: Scientific Computing Using Python, because I used the print command.

So today we will stop here. You can save this notebook and give it a name. I have named mine "lecture_1". You can name it as you like and save it in your own folder, maybe named Python Code for Scientific Computing. You can even save it on your desktop and name it "lecture_1" or something similar.

(Refer slide time: 52:08)


localhost:8888/notebooks/Desktop/PythonCodsVivek/Lect1.ipynb
jupyter Lect1 Last Checkpoint: 8 seconds ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Rename File
File Path
Lect1.ipynb
New Name
Lect1.ipynb
Cancel
Rename
NPTEL

```
[41]: # Illegal variable names
#5myvar = 'Not OK'
#my-var = 'This is also not OK'
my var = ' Not ok'

  Cell In[41], line 4
    my var = ' Not ok'
      ^
SyntaxError: invalid syntax
```

Various forms of variable names (multi words)

```
[42]: thisCourseName = 'Scientific Computing using pythor

[43]: ThisCourseName = 'Scientific Computing using pythor

[44]: this_course_name = 'Scientific Computing using pyth

[45]: print(this_course_name)

Scientific Computing using python

[ ]:
```

So today we will stop here. In the next lecture, we will continue. Today, we just discussed the basics of Python—how to define variables, how to compile them, which compiler we are using. We have discussed all these things.

In the next lecture, we will discuss Python commands. So thank you all. Namaste.