Scientific Computing Using Python

Professor. Vivek Aggarwal and Professor. Mani Mehra

Department of Mathematics

Indian Institute of Technology, Delhi

Lecture No. 02

Hello, welcome to Scientific Computing Using Python. In the last lecture, we discussed some basic commands of Python, so today we will start from that.

In the last lecture, we discussed variable names. We talked about camel case, Pascal case, and snake case. So today, we will start from that. This was our lecture one.

Now, I will open another file like this, and I will name it as lecture two by rename it. In this, I will write lecture two, and from here, I will use markdown and press Shift + Enter. So, this is our lecture two.

Today, we will go a little further and see what values we are giving to the variables. So far, we have given one value to each variable. Now, what we can do is assign values to multiple variables at once, and that is called multiple variable assignment.

As an example, I have defined the variables x, y, and z. I have defined these three variables, and in this, I will take a string. Suppose I define "Apple", then "Banana"—okay, I am taking strings—then I take a comma, and then I take "Pineapple".

Okay, so what have I done? I have assigned "Apple", "Banana", and "Pineapple" to the variables x, y, and root in a single command. After that, I want to see how this works means whether its correct or not.

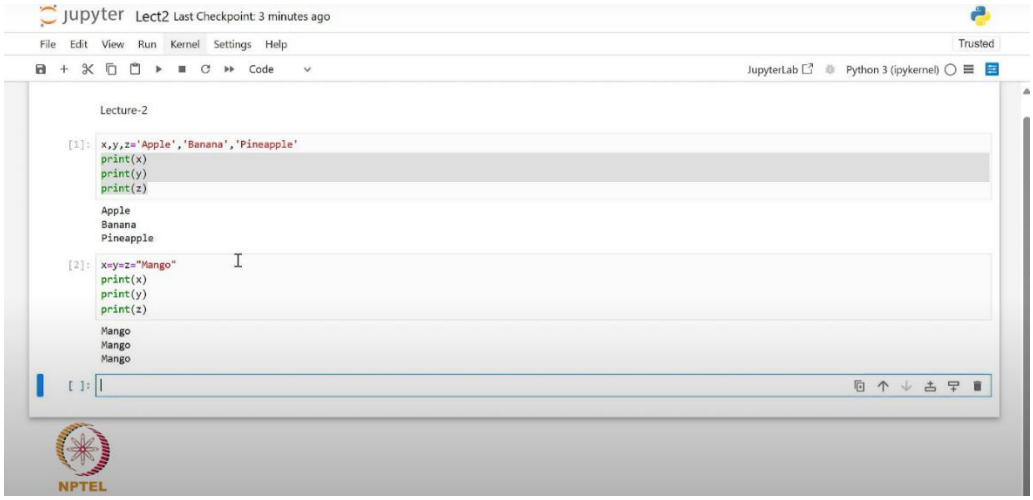
So, what do I do? I print it. I print (x), print(y), and print(z). I ran it by pressing Shift + Enter. So, I see here: "Apple", "Banana", "Pineapple". These three values are printed.

So, in this way, I can assign values to the variables x, y, and z in a single command line with different values.

I can also do this: if I have three variables and I want to assign the same value to all three, I can do that as well. Suppose I write x=y=z="Mango", and after that, I print it. I copy the print command and run it. I assigned "Mango" to all three variables at once. In this case, the values of all three were the same.

So, I assigned "Mango" and did this. What we did was that we defined the variables x, y, and z, and now we have values.

(Refer slide time: 4:35)



If we take any value, any vector, then we keep it in the form of a list. Now, if I make a list, for example, I made a list called fruits. I named it fruits. I wrote "apple" as the first item. Okay, what should I write as the second? Let's write "mango", and in the third, let's write "banana".

So, we have fruits as a variable name that has the values "apple", "mango", and "banana" as fruits= ['apple', 'mango', 'banana']. Creating it means we have defined a vector.

If you see, the bracket is a square bracket, so the meaning of square brackets is—now if you remember—we use such brackets while showing matrices also. So, this means that this vector has been defined. The first element is "apple", the second is "mango", and the third is "banana". So, Fruits is now a vector quantity which has three values.

Okay, now what did I do? I want to define variables using this list. So, I defined the variables x, y, z = fruits.

Now see, how many items do we have in fruits? Three. And how many variables do we have to define? x, y, and z—we have to make it three. So, from here, I took the values of x, y, and z from fruits. After that, I printed x, y, and z, and I pressed Shift + Enter. The value that we got was apple, mango and banana. So, what does it mean? It means that we have defined the variables automatically, directly from the list. This can be done using this process called unpacking.

Suppose I am in the third cell, and I had to write something before this. Then I can go here, click Insert cell above. So, if I click here, it comes above, and I will give it the title Unpack a Collection

We have a collection; we have to unpack it. So, I define for myself what my next command is. So, this is unpacking the collection. And this is not code—we will just markdown it, and I will enter it here.

So, from here, it is known that the next command is the unpacking of the collection. We had a collection: "apple", "mango", "banana"—and we unpacked it. From there, we defined variables and print it.

Similarly, I took another variable. I took this variable—I took fruits only. Okay, but now what did I do? I wrote x, y = fruits. Okay, I wrote this and defined fruits. After that, we also

pressed Control + Enter, and see—there comes an error: too many values to unpack. Because the values we had were three, but we defined only two variables. So here, it will give: too many values to unpack. It means there are many values inside it which are unpacked, so this command will not work, and it will give an error.

So, we can define as many variables as the number of values we have in the vector. If we define fewer variables, then we will get errors like this. So, we can do these things. Like this, we can define this variable with the help of listing.

(Refer slide time: 8:30)

Now, let's go a little further. If we want to see the output of something, how can we see the output of the values? We know that we can see the output using print. So now, I write the output variable. I want to define it. So, I markdown it and pressed Enter.

Now see, what are output variables? They are variables through which you can see something on the screen. It is called an output variable, or simply output. Like if it is print, then print—we know that whatever I write, it gets printed here. So, we will not know like this. This print that we have is an output variable—a function defined.

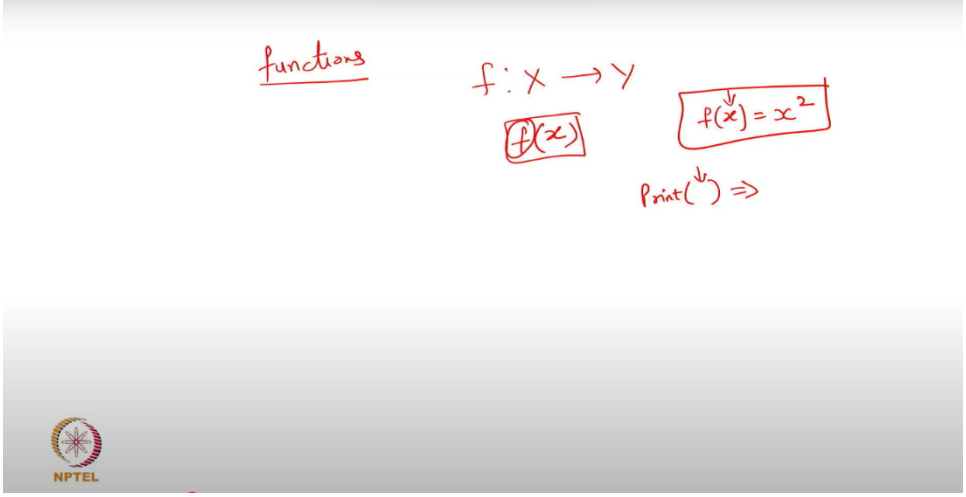
So, wherever we print in this way, what is it doing? If we write print—just wait a second—I will tell you. This is how it is. Now, in this way, the variable that we have—what do we do with the variable? How are we defining it?

If you see, as we used to do in maths—in maths, we used to define functions. What are functions? We define f from some set to some set. We mean domain and range. Suppose we do not have knowledge of that. Suppose this is a set X and Y. We used to define a function, so we used to write the function like this—always with a bracket inside it.

Suppose we take any value from here, an element from a set—we took small x—and the values went into the space here. So, we used to write the function like this: We have a function f(x) and suppose I define the value of this function as x^2. So, any value that comes here in this function will be squared, and we will get the output.

In this way, we define all the functions in Python. Like we have print—print is also a command. If you see, it is a function. Whatever variable we give inside it; it will bring it to the screen.

(Refer slide time: 11:50)



So, we define variables and functions in this way. We will further define all the functions in Python. But we have to keep in mind that these functions are different. We will take different functions and input variables inside them, and their outputs will keep changing.

I have defined an output function like this. Here, we have printed—the word "apple" will appear.

So, what we have to do now is see how we can write anything through print.

Like, I have defined a new variable: x = 'Python is good' and then print it: print(x). So, what did we do? The value of x should be printed. And yes—it got printed.

Like this, I defined the variable: x = 'My'. Then I defined a new variable and wrote: y = 'Name' and z = 'is Ram'. So, I defined this. After that, what do I do? I print it. Now I want to see if I can do it in one go. So, I wrote: print(x, y, z).

I had defined three variables. x has only 'My' string in it, y has 'Name', and z has 'is Ram'. I printed them, and the output was: My name is Ram

So, what did we do in this print command?

There were three variables—we entered them as inputs. So, what happened? print took all these three inputs and displayed the values inside them on our screen.

We used the print function like this.

I can write this because all these three are strings.

So, I defined the same and tried to write it like this: print(x + y + z)

Let me see what happens.

Because what's happening is: x is 'My', y is 'name', z is ' is Ram'.

But when I printed them we got MyNameis Ram, the output had no spacing between "My" and "name". So, to fix this, what do I do?

print(x + y + z). Now what's the problem? Again the same problem . So, what to do for this. We need to change a bit in variable. So, we added spaces manually between the strings. Now when we print(x+y+z), the output is:

My name is Ram

(Refer slide time: 15:22)

So in this way, we print values. We did x, y, z, or else we concatenated them using + and joined them together.

Suppose now I wrote a string x:

x = ‘Name’ , y = 5 and then print(x + y)

Now x is a string and y is an integer. What is the problem?

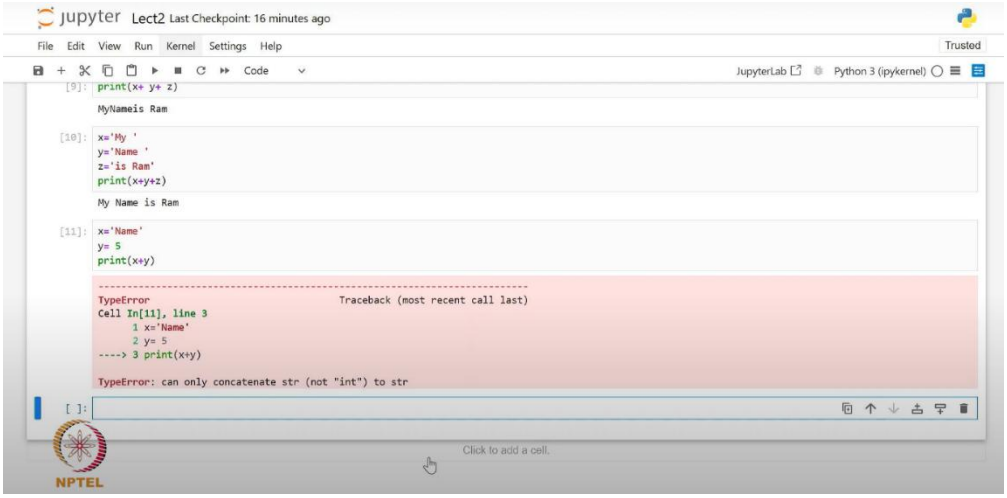
Python says: can only concatenate str (not "int") to str

That means it can only add two strings. It cannot add an integer value and a string together.

So if we are adding two variables using +, then their data types must be the same. If one is a string, the other should be a string. If one is an integer, the other should also be an integer—or at least be converted to the same type.

Otherwise, it will give an error.

(Refer slide time: 17:30)



So in this case, if I want to print variables of different types together, how do I do that? Let's say: x = 'Name' and y = 5. If I do: print(x, y). It works. The output will be: Name 5. So we can print like this—but we cannot use + unless both are the same type. We can only add them using + when they have the same data type—either both are strings or both are numbers. This is how we use all the variables.

Like here, I had written x. Earlier, I had defined x = 'My'. Then I redefined x = 'Name'. So now 'Name' has replaced 'My'. This is called variable reassignment. So, all these variables we are using are called local variables.

So, similarly, we can define some variables in the form of global variables as well. Now, we will write about global variables. In the program that we have, there are some local variables and some global variables.

A local variable is like, if we take any if command in that location (e.g., looping), then the variable that we define in that loop is called a local variable. But a global variable is one that, once defined, can be used in every part of the program. Its definition remains the same everywhere. We call this a global variable.

So, how do we define a global variable? Let's see what the meaning of a global variable is.

For example, I wrote x = 'beautiful'. I defined a string.
After that, I defined a function. So how do we define a function? Using def .

We can name it myfunc, as per the slide we just showed you.

```
def myfunc():
    print('Python is ' + x)
```

So, I defined the function and wrote print('Python is ' + x). Now, x is a string and 'Python is' is also a string. After that, I defined the function and ran myfunc(). I pressed enter, and the output was "Python isbeautiful".

If we want to add spacing, we can just add a space after "Python is " inside the string. Now, what does this mean? That x was a variable whose value we defined as "beautiful".
After that, we went inside the function, and that value did not change. The function used the

same value and printed "Python is beautiful", meaning it worked like a global variable. So, if we define a variable outside a function and then use it inside a function, it behaves like a global variable.

Now, suppose I write this:

x = 'beautiful'

def myfunc():

    x = 'fantastic'

    print('Python is ' + x)

myfunc()

print('python is ' + x)

Here, x = 'beautiful' is defined globally. Then, inside the function, we defined another x = 'fantastic'.
When we run myfunc(), it prints "Python is fantastic". But when we run print('Python is ' + x) outside the function, it prints "Python is beautiful".

This means the variable x defined inside the function is a local variable, and it is only being used locally. The outer x remains unchanged and is used elsewhere.
So, the local variable means it is being used only locally inside the function; it is not affecting the outside variable.

(Refer slide time: 24:52)

Now, suppose I want to define a global variable inside the function. Let's see what happens:

def myfunc():

    global x

    x = 'Good'

myfunc()

print(‘Python is ‘ + x)

Here, we defined a variable x inside the function and declared it as global.
After the function call, we printed ‘Python is ‘ + x, and the output was "Python is Good".

What does this mean? The variable x, which was defined inside the function, was actually a global variable due to the global keyword. So, after coming out of the function, if we print x, it still holds the value "Good", which we defined inside the function. Hence, this variable x is now treated as a global variable throughout the program.

Let’s take another example:

x = ‘beautiful’

def myfunc():

    global x

    x = ‘fantastic’

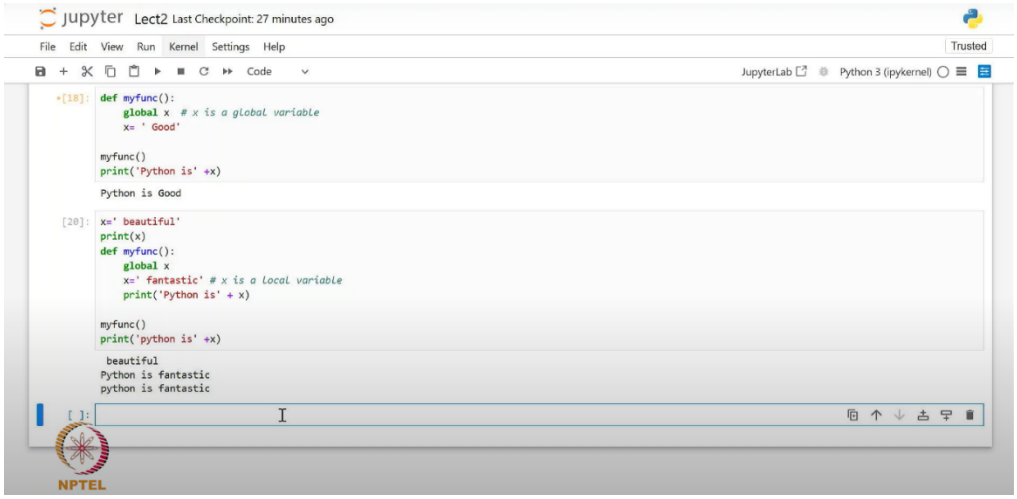
    print(‘Python is ‘ + x)

myfunc()

print(‘Python is ‘ + x)

Initially, we defined x = ‘beautiful’. After that, inside the function, we made x a global variable and set its value to ‘fantastic’.
Then, we printed it both inside and outside the function. The result in both cases was "Python is fantastic". This means, as soon as we define a variable as global, its value changes globally. Wherever we use it afterward, it will reflect the updated global value. So, in this example, the global variable was x = ‘fantastic’, and the program used that value consistently, both inside and outside the function.

(Refer slide time: 28:45)



Now, the next thing we do is data type. Data type means how many types of data do we have. We know that our data is string, our data is integer value, our data is floating value, and our data is character. So, how do we define different types of data? It comes in the form of data type.

Now, like we have x, I took the value of 10. Okay, and printed it. It came out to be 10. Now, we don't know what type of value it is. Now, I wrote x like this. I wrote y inside these commas as y = ' 10 ' . And we know that if we write inverted commas inside it, it becomes a string. But now, let's see what will be written. print(y). It is also written as 10.

Now, we have written x as 10 and y also got the value of 10. But how will we know that the value is 10? Because we have the same value, so what will we do now? We will find out their type. So, we do print type.

Now, we had a print function, such a type function; the name of the type function is type. If you give any value inside it, it will output the type. Like, we had the function f(x) = x^2, so if we put x there, then x^2 will come out. Okay, if we put 4 value inside it, then 16 will come out. Similarly, we have a type function, so if we put any variable in the type, it will tell you what type of function it is. So, like I typed: print(type(x)). I defined it here. Like this, I typed print(type(y) and entered it. Then this came up.

Now see, first came <class 'int' >. So, in the first, the value of x was integer value 10. Second came string. Why? Because we had defined y with this type in inverted commas. So, if we define it in inverted commas, then it will take as character and it will come in the form of string. So, this will show us the type of our variable if we don't know.

(Refer slide time: 31:42)

Similarly, we have a new variable z = 2.3. I will write this, okay, and after that, I will print it: print(z) It came up with 2.3 . Now, we have a new variable and we don't know its type, then I will also write its type as print(type(z)). It came out as < class 'float' >. So, what does it mean? z is a floating-point number. z is a fraction 2.3, okay? So, whatever comes in decimal, we call it floating point. So, this is floating point, floating number came up.

Like this, I define new variable xx and define it ‘Good’ , I wrote it like this, okay. I print it xx, I print it, comma here, type(xx) as print(xx , type(xx))

I define a new variable. Let's do this and see what happens. So, this is written: Good <class ‘str’ > . What is xx? Good, and what is its class? String. So, I printed both the values in a single print, which we had done earlier also. So, this string came.

Similarly, I take z=2+3j . Now we know what 2+3j would mean. We write i for complex number but we do not have to write it in the form of i. In this, we have to write it in the form of j. So, this came, okay. I printed it and its type. I wrote print(z, type(z)) . The value is 2+3j and its class is ‘complex’. From the type, you will know what type of form it is.

Similarly, we can define a list. List, as I told you earlier, okay? So, in the list, now we have written like this: Like we define a variable we have defined a list. So, in the list, I have named it suppose name, okay, and I have written time. I will give another value, let's give some value like age. Okay, so I have defined it as :

List=[‘name’ , ‘time’ ,’age’]

So, the name, time, and age—I have defined a listing.

And after that, what do I do? I will print it, the type of the list. I have written its name. I will define its type, okay. And let's see what happens. So here it comes <class ‘list’ > , it means its class is a list.

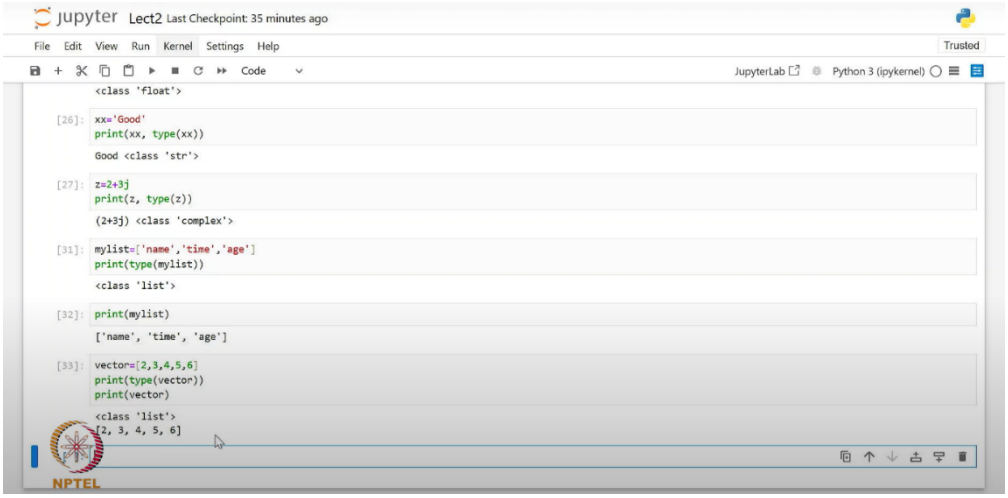
It is not that we should keep only list; we can name it like this, mylist . I have named it mylist. Now, I have defined it. Now, I write print(mylist) here. I have defined a list, print it. It comes [‘name’ , ‘time’ , ‘age’]

This listing is only when we have a lot of variables, we have kept them in one variable. If we want to define it in vector form, then we know that we have to write it in vector form. We have to write it in matrix form. If its dimension increases, then we have to write it in matrix form or vector form. So, for that, we have to define a list.

I can define it like this. Now, we have defined a list. I can define a vector like this using a string variable. So, I have defined a vector =[2, 3, 4, 5, 6]. I have defined it like this. So, we have a vector defined.

Now, I wrote this: print(type(vector)) like this and I pressed enter to write print(vector). So, now, what did we do in this? What is a vector? It is a list. So, this is a listing. And when we printed it, it had values, then 5, and these were its values. So, we can define the value like this. So, we have a vector.

(Refer slide time: 36:42)



Similarly, we had a listing. So, we can define it like this. Now, if I take a vector, let's see, if I take another vector. Now, it is not necessary that only these numbers have to be taken. I can define it like this also.

Now, like I have defined a vector there with square brackets, so you see that if we do not write the square brackets and I write this here we did not take the square bracket like vec=(2,3,4). So, let's see what happens. Let's define one more vector : vec1. Inside this also, I did not take the square bucket. So, inside this, I define, what should I do? Let's take vec1=('scientific' , 'computing' , 'using Python') , I wrote it like this, okay, I defined this. Let's give some spacing.

So, I defined a vector vec and assign it values 2, 3, 4 and another vector : vec1 .Now, let's What is its type? So, I wrote print(type(vector)). Similarly, let's see the second one, print(type(vec1)) defined this, pressed enter.

class 'tuple' is written. Earlier it was written list. So, what does it mean? List will come when we write square brackets and otherwise, if we write simple brackets, tuple will come in it.

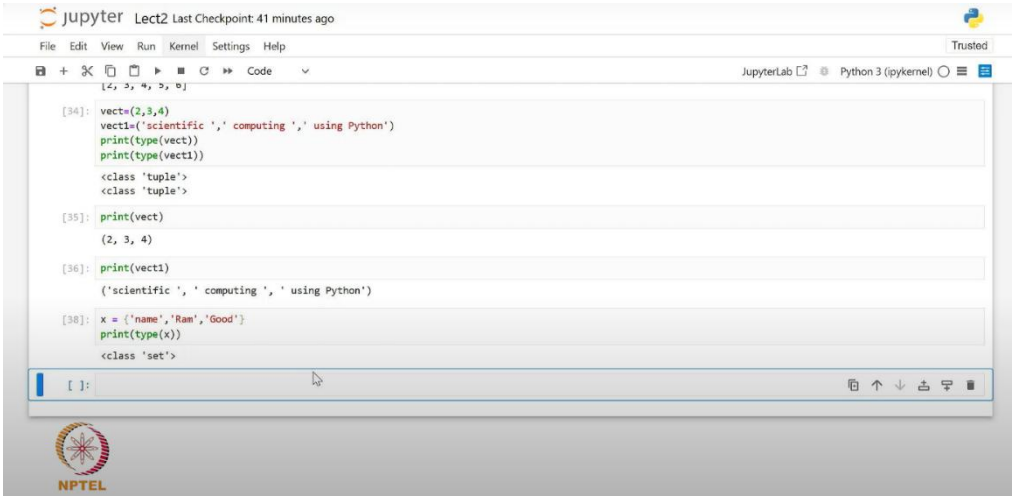
So, the meaning of tuple is, we know what a tuple is. So, a tuple is basically a set, which is a sequence of numbers or anything. Now, like vect is in the order 2, 3 ,4, so we ordered it. So, its name came out as tuple

Similarly, we wrote vec1 in ordering, scientific computing using Python. So, if you put a number or a string in any order, then we call it a tuple, okay? Now, we can also print the values of these tuples. For example, print(vector) will print the tuple values. Print basically tells us what is inside it, what will be its output. Now, if we go ahead and do more programming, then we will see the output in it; otherwise, we will print, and only after that will we know what value will come. Okay, so here we printed its value, so we call this a tuple.

Now, let's see it in another way. We know that we use brackets in three ways: one is curly brackets, one is simple brackets, and the other is square brackets. I have used two of them. Let me use curly brackets. I defined a variable x there. I named it x = {}. I took curly brackets. Inside curly brackets, I wrote 'name'. Okay, after that, comma. Then, I wrote 'Ram'— you can write anything. 'Good', I wrote this. Okay, I put curly brackets around it.

Now, we have to check what is its type. So we write print(type(x)) so that we get to know. We can give some spacing to x so that it becomes more clear. And we printed the type. So now let's see. This is what is written — this is a set. So what is this? x is written as a set. So it means this is a set. Earlier it was a list, then it became a tuple, now this has become a set. So what is a set? It also has ordered values, okay? So we defined the values in it: 'name', 'Ram', 'Good' . So we got these values as set.

(Refer slide time: 42:50)



Now, let's take one more. I defined y . Let's take the same. So I defined the same ,I applied a colon and changed it a little. I wrote the 'Name', colon, after that I defined ' Ram', then put a comma, okay? After that, I defined 'Good' then 'Age', then applied a colon, and wrote age: 35. You can take any name. So I took the name 'Rakesh' . I defined it by putting a colon. Now I wrote this and put it in curly brackets. In curly brackets, it means this is a set. After that, I printed it, print(type(y)). Now it is written class 'dict'. Dict means dictionary. So the dictionary is now created. In the dictionary, we have defined the name as Rakesh and age as 35. So we have defined this dictionary.

Print now, let me see what comes out. After printing it, it is written that the name is Rakesh and his Age is 35. So, like this, we have defined sets. If you define it by putting a colon in it, then the dictionary gets created. So listing, tuple, set, dictionary — all these, by changing a little bit, their definition gets changed. So you have to keep this thing in mind a little bit.

Now, we have defined ours. Now, like we are defining numbers — how many types of numbers will there be? How will we define them? So like this, we can define in the form of numbers. So let's write numbers here, Python Number system — I wrote it, I marked down, and defined it.

Now, see how we define it. We know that we have defined numbers. Now, like I wrote x = 2.345. Suppose I print it and write its type. Then it will be written float. So float means this number is a floating point number. Above, we are writing x. Many times, we have defined many x's. Somewhere, we have written x as an integer; somewhere, we have written x as a string. But it shows only the updated values. So we have defined x as a variable in the computer whose value we keep changing. Sometimes, we are writing it as a string; sometimes, we are writing it as a number, okay?

Let's see now how can we define numbers in Python. So I have a value int. Now, like I wrote x = int(). int is a function defined — whatever value comes inside it, it will give its integer value. So like I wrote three here and then I did print(x). I wrote it and along with that, I also write its type. I entered and it came out written, its value and class 'int'. The type means what type of variable is x — that is, an integer variable, it is in integer form. So x is integer and its value is in integer form.

Now, if I write the same thing again — if I write x = 2.3 and print it, we have already write this — let's write int(), okay. I went to int() and I wrote y= int(2.3) . After that, I printed it. So I wrote print(x, type(x))) , so, the number and the type of it came out. So now, see what is written — what is the value of x. It came out just 2. It was not 2.3; it came 2. Why? Because we defined it as integer, and what is its integer value? It is 2, okay? Its integer value is 2 and defined it as class int.

(Refer slide time: 48:59)

In this way, if I define a new variable as float, suppose its value is 2.35 or I have taken 2.3456 instead of 2.3, something like this in decimal, and I have printed it. What will it print? I have also asked its type by print(y,type(y)) . So this is written: the value is 2.3456 and float is there.

But if I have to write the same thing again, if I copy-paste it and write its value as 2, then we know that this is an integer. There is no fraction in it, there is no decimal. But I have defined it as float. So like I defined float and pressed enter, then its value will be 2.0. So automatically, it has made zero a decimal. So 2.0 came and its type is floating. We already know that.

So whatever function we define, it is an integer function, it is a float function. In such a case, I define a new variable x as equal to string, and I take it str(3). So I defined a new function and I wrote it as string. I then defined y, wrote it as string and wrote "Python" in it. Then, I defined z as a string and write 2.3 in it. I wrote okay.

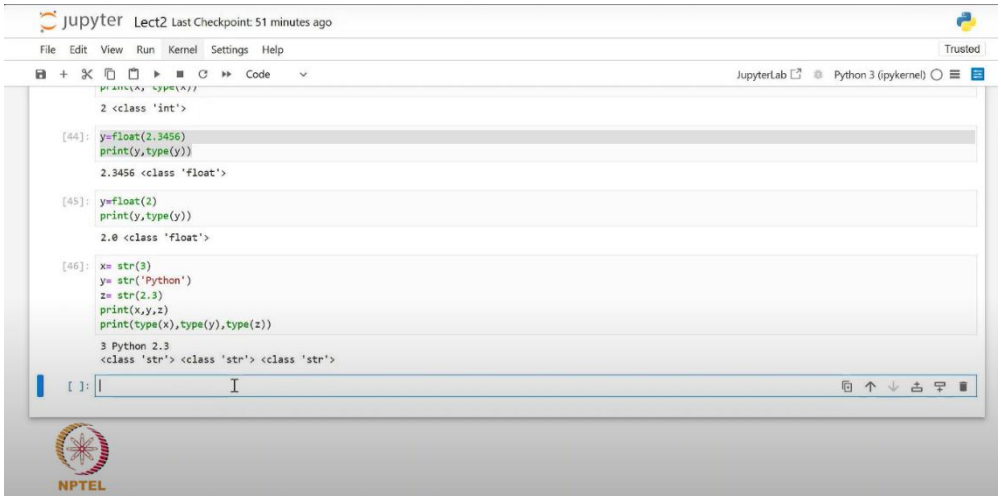
So what did I do now? I defined all three variables as string. I printed them with print(x,y,z) and wrote print (type(x), type(y) , type(z)) and then enter. It is written: 3 Python, 2.3, and what is its class? Look, string, string, string. So it means that these three are no longer integer values. They are working like strings. Python is already a string (a character). Now 2.3 is also

working like a string. Now ultimately, what we had —Now we have 2.3, which has been defined as a string; the value has been defined as a string.

So we have this type of data. We can define what is integer value, what is floating value, what is string value, what is integer value. When we program in a computer, we can define variables. So we have defined it like this. Now we will have all these things.

So now, let's see how to define a function. What is an inbuilt function in Python? So first, we will see the functions of Python. We will also check Python functions, inbuilt functions, how do we define them? So now, we will define inbuilt functions.

(Refer slide time: 52:29)



Now, if we have a function, as you must have seen, which is well defined, we have a function like absolute value. So abs() — this is a function. Now, the function has to be given some input, so suppose we have given minus 2 as the input. Now, we have defined absolute as a function. Like in math, we defined mod x. So mod x is a function. We have defined a new function. So what does mod x mean? Let me write this: It means absolute value. Now I entered it and it was written 2.

So its absolute value, which is magnitude,it will give. Now we can apply the same thing in it. Now I wrote absolute value and I defined it as 2 + 3j. So it's a complex number and we know its absolute value is x squared plus y squared and the square root of this. We should have it like this. So this value is 2 squared that is 4, 3 squared 9 , so under root 13. So the value we got is of root 13. So we will have it defined like this. Now, if we want to take the absolute value of any number, it can be complex or negative, if positive its value will remain same.

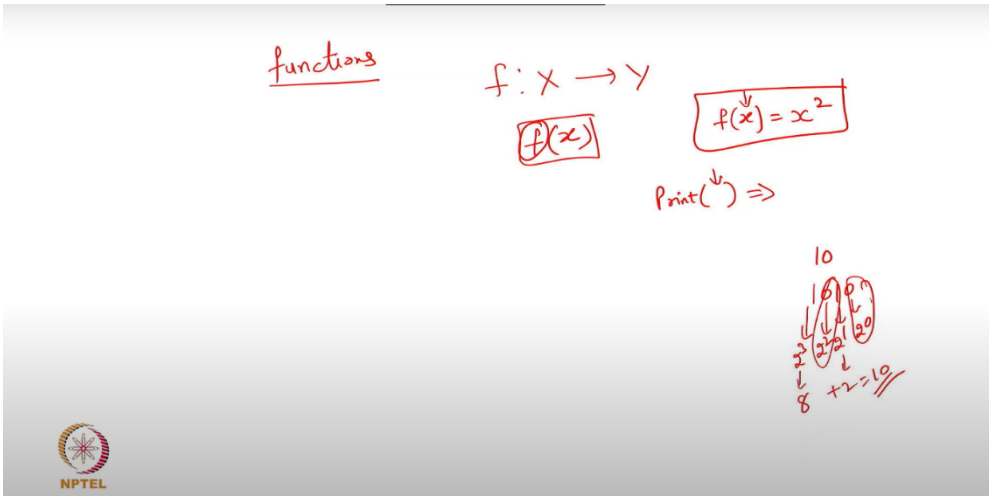
We can define another function like this. When we do this course, we will discuss binary numbers. So, if we want to convert any number into a binary number, then there is a well-defined function and its name is bin() . bin means binary.

Now suppose I want to define 10 as binary number. So, I write bin(10), hit enter and it came up. So, 0b means binary, and the value of n is 10 (decimal), because the value of 10 we know how it is defined; we can write it like this.

So now, for example, if it is 10, then its value is 1010. So this is the power of 2 to the power of zero, power of one, 2 to the power of 2. So this is the zero value, this is also one. So what

does it mean? 2 to the power 3 is 8, and this is 2 ; this becomes zero, zero into this. So 8+2=10 would be the value if it was defined like this.

(Refer slide time: 56:58)



So what is it telling us? That whatever number we take, we will get its binary representation immediately. Okay, we got the binary number.

After that, if we need a big number like bin(100), then this is what we get(0b1100100). Yes, this is the binary representation of 100. So this is the binary representation of any number. It is the binary representation of 100. So we can represent binary numbers in this way.

We have different types of functions — well-defined functions. So ASCII is also a function which you must have heard. So, it is of this type of form. So ASCII form is a representation of the American standard of this type.

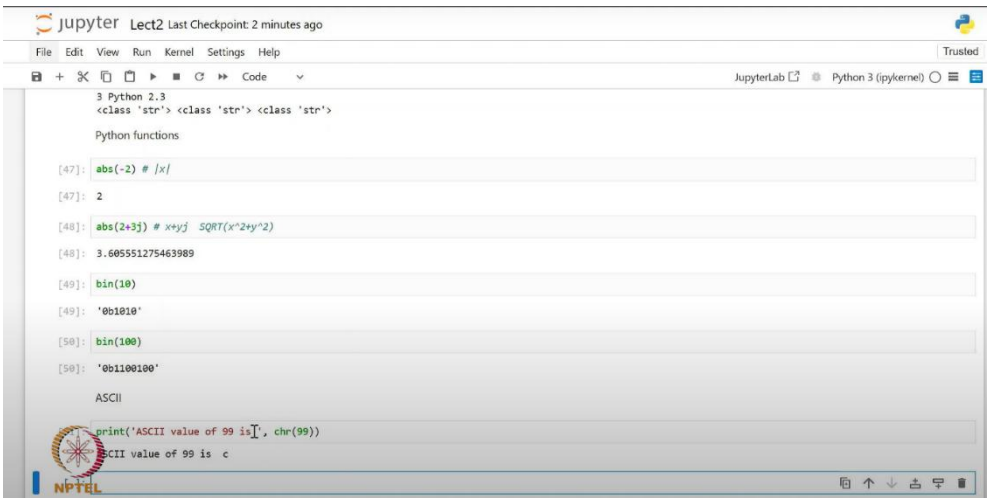
So what is that representation? If I write any value, let's print it. Let me write:

print('ASCII value of 99 is ' , char(99))

So, it is written ASCII value of 99 is c.

What does it mean? The ASCII value of 99 is c.

(Refer slide time: 59:20)



What is chr? Character is also a function like — absolute we used , similarly character is also a function like integer, string, float. So, in such a case, 99 became a character, and its value of chr(99) is 'c'.

So now, for example, if we write it, I will write print(chr(80), chr(121), chr(116), chr(104), chr(111), chr(110)).

Okay, let's print it and see what comes out. When we printed it, it was written P y t h o n.

So these ASCII codes have a coding, so according to that coding, chr(80) which is P represents 80, or chr(121) represents y. There are ASCII codes like this; according to that, we can show their value. So now we have the values that we have. Okay, so Python written in this way.

(Refer slide time: 1:01:08)

Let's do this till here. so, these are all the commands that we did today.

So, keep these commands in mind because in the future we will program and use scientific computing, so all these commands will be used there.

So, I hope that this lecture will be very beneficial for you, and we will do future programs using these commands.

So, thank you for this lecture on scientific computing using Python.