

Numerical Analysis
Prof. S. Baskar
Department of Mathematics
Indian Institute of Technology-Bombay

Lecture-08
Introduction to Python Coding

Hi. In this lecture, we will quickly have an introduction to Python coding.

(Refer Slide Time: 00:23)



We will use Python language to develop codes for some numerical methods that we will develop in our course. The first question is why Python? We have so many languages. One of the main reasons for, why we choose Python is that it is an open source programming language means Python is available for free. In fact, many operating systems come with Python pre-loaded.

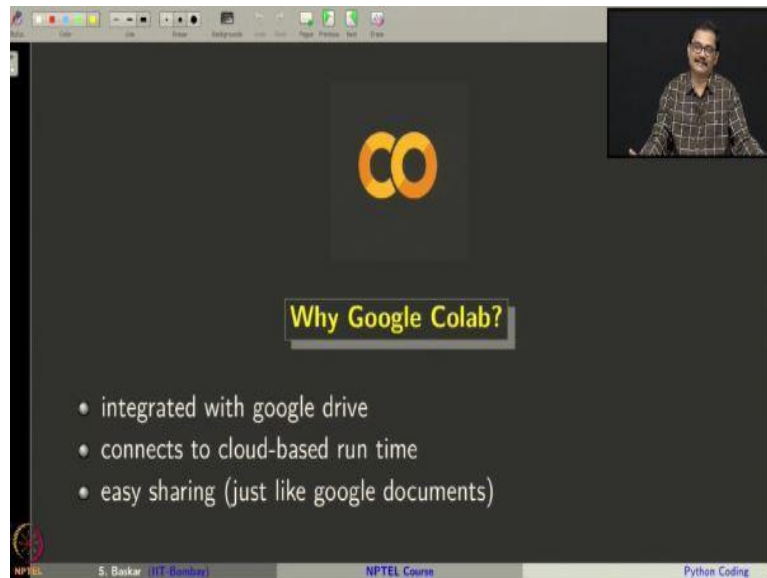
The second reason for, why we want Python is that it is a cross platform language, means you develop your code in one platform say on windows and then takes the code and run it on another operating system say Linux. You can do it without making any changes in syntax or any setting. That is what is meant by cross platform language. In fact, it has many inbuilt libraries that make the programming more simple.

Especially, when we are developing some complicated softwares are working on coding some complicated scientific problems. Python is a very popular language it is used as a tool in many famous websites. And therefore, python has a large user network; this allows us to get some

supports like whenever we have problem or doubt you can go to some discussion forum and pose your questions.

It is very likely that you will get reply. In that way we can say that we can get free support when working with Python. So, these are the reasons for why we preferred Python as a programming language to learn how to code certain numerical methods.

(Refer Slide Time: 02:36)



Having said this, the next question is what is the editor that we want to use to write Python programs and execute them? Well, there are many editors like visual studio or anaconda is there which has a collection of editors and different Python distributions, but in our course we prefer to use Colab. Colab is a Google product and again the question is why we want to use Colab when we have many other editors?

One important reason is Google Colab is integrated with Google drive, means whatever program that you write and save will be saved on your Google drive. This is very comfortable for us. The next thing is Google Colab connects to cloud-based running time, it means when you execute a Python code on Colab it connects to a cloud-based machine and executes your program on the cloud machine.

This is particularly very nice, because you do not need to install any software or libraries on your personal computer, because your code is basically going to be run on a cloud machine. And Google Colab has almost all the commonly used libraries already installed in it. In case,

if you want to use a library which is not found on Colab you can install it on your Google drive separately.

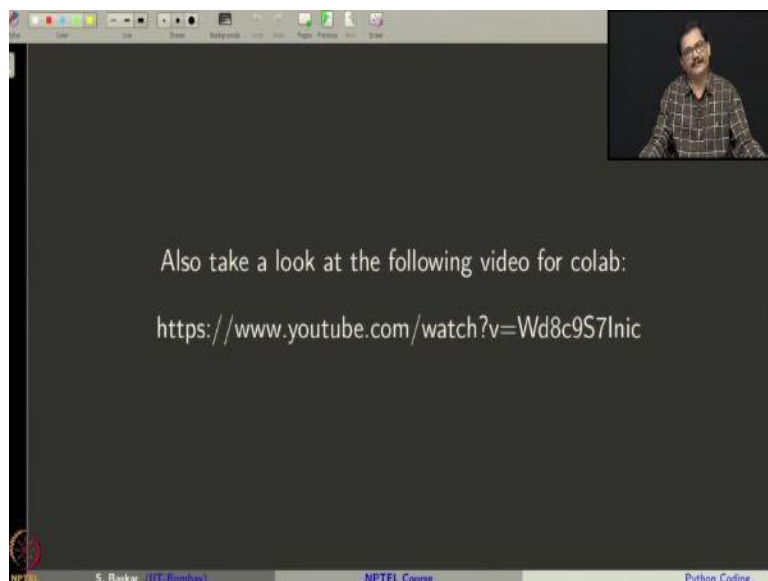
In that way, you do not need to install any extra software or libraries on your personal machine. And of course since all your programs are saved on Google drive it is easy for you to share your programs with your colleagues or friends, just like how you share Google documents.

(Refer Slide Time: 04:46)



And these are the reasons why we want to use Python as a programming language to learn coding of some basic numerical methods and we want to use Colab as our editor.

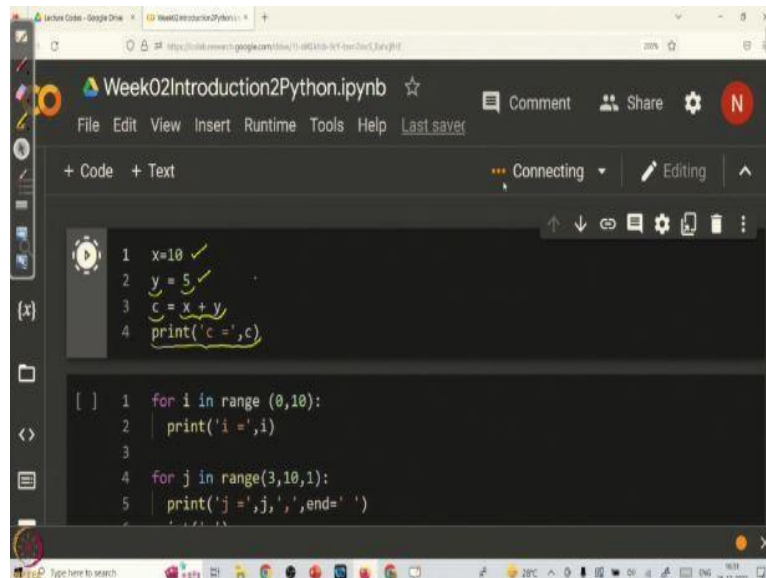
(Refer Slide Time: 05:04)



I will not be introducing Colab, because it is very easy for you to understand how to start with Colab and how to use Colab as an editor. You can also find many videos on YouTube to learn

Colab within 5 to 10 minutes. In fact, I have also made a short video on how to start with Colab, you can find my video on the links shown here. I will also give this link in the description box. With this we will directly go into the Python programming.

(Refer Slide Time: 05:43)

A screenshot of a Google Colab notebook interface. The browser address bar shows a URL starting with 'https://colab.research.google.com/...'. The notebook title is 'Week02Introduction2Python.ipynb'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu, there are tabs for '+ Code' and '+ Text'. The code editor shows two code cells. The first cell contains:

```
1 x=10
2 y = 5
3 c = x + y
4 print('c =',c)
```

The second cell contains:

```
1 for i in range(0,10):
2     print('i =',i)
3
4 for j in range(3,10,1):
5     print('j =',j,',',end=' ')
```

Well, Python is a easily readable programming language, it is not very complicated. Therefore, I will quickly go through certain commonly used tools from Python in our course. The first thing is how to define a variable, store a value in the variable and how to do certain arithmetic operations and print the output? A variable x is defined here and the value 10 is stored in the variable x. For that we have to write like this.

Similarly, $y = 5$ makes the Python to create a variable y and store the value 5 in that variable. Then what we are doing is we are defining another variable c and we are storing the value of the sum of $x + y$ in that variable c and finally we are printing the value stored in the variable c. Let us see how to run this program. **(Video Starts: 06:57)** To run the program we just have to click this play button for the first time the Google Colab will take some time to allocate some memory in the cloud machine.

The next time onwards it will run your code more faster. Well, it has established a connection with the cloud machine and it run your small bit of code and it has shown the output as $c = 15$.

(Video Ends: 07:24)

(Refer Slide Time: 07:26)

The screenshot shows a Jupyter Notebook window titled "Week02Introduction2Python.ipynb". The code cell contains the following Python code with handwritten annotations in yellow:

```
[1] c = 15
```

```
(x) 1 for i in range(0,10): → 0, 1, 2, ..., 9
2   print('i =', i)      i=0
3
4 for j in range(3,10):  j=3, j=4, ..., j=9
5   print('j =', j, ', ', end=' ')
6   print(' ')
7 for k in range(0,10,2):
8   print('k =', k)
```

The code is executed, and the status bar at the bottom indicates "0s completed at 4:34 PM".

Next, let us see how to run a loop. There are 2 ways that you can create a loop. One is using the for loop and another one is while loop. In this program, we will see how to use for loop. The syntax for for loop is for and then running index and its range given here. You can specify the range in any way. In Python you have a special command called range which will create a sequence of numbers ranging from 0, 1, 2 up to one number less than what you specify here.

This is very important to remember, it will not go up to 10 just because we give 10 here it goes one number less than that. And then it takes the value $i = 0$ and then executes all the lines written below the for loop with an indent here. You can see that there is an indent given here. So, whatever line you write below this with an indent will be group executed under this for loop and it will go from $i = 0$ till 9.

Let us run this program and see what is the output for the first for loop. Similarly, there are also other two for loops, which we will see little later. Let us now concentrate only on the first for loop. **(Video Starts: 09:09)** In fact we can comment all these lines by just selecting them and pressing command slash. So, in Python if you type a line which starts with a hash then Python will not execute those lines.

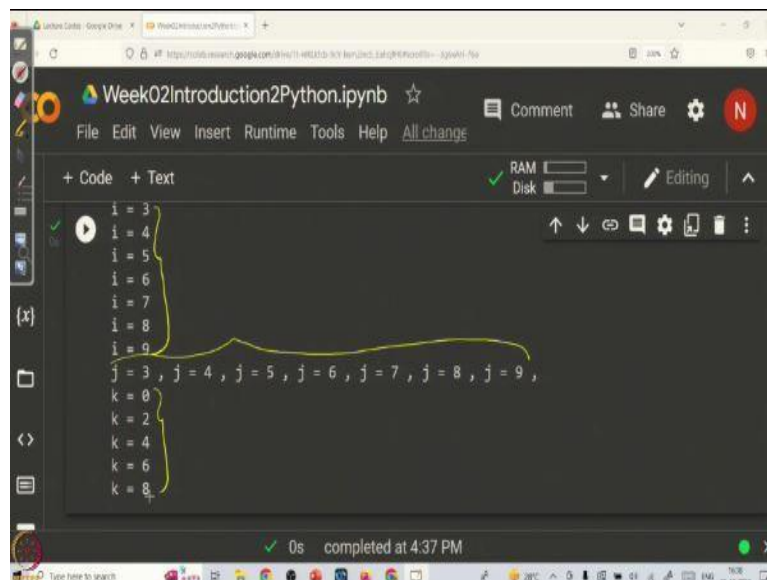
So, here Python will not execute all these lines. Let us run this code, where it will only run the first for loop as we expected you can see that it starts with $i = 0$ and every time we are printing the value of i . Therefore, you can see that $i = 0$ up to 9 only it went. That is very important as I shown here it goes from 0 to 9 not up to 10 as we have written here. That is one thing that we have to remember. **(Video Ends: 10:07)**

Next let us take another for loop where we are going from 3 to 10. Range has basically three arguments. The first argument tells you, where your range starts and the second argument says where it ends? Actually, it ends one number less than what you write and the third argument is about the step in which the numbers are incremented. If you do not specify anything as a third argument then by default range takes it as 1.

Therefore, here writing comma 1 is equivalent to not writing it at all. So, in fact you do not need to write it at all, if it is just incremented by 1. You can see that the second for loop will print $j = 3, 4, 5$ up to 9 again, because you have given 10 here therefore it goes up to 9 and what about this? This will make the print to be $j = 3$ comma and then it we not go to the next line, whereas here the outputs are printed one below the other.

Now it will be written one after the other it is something like this up to $j = 9$. And then this print command will put a carriage return and comes to the next line to print and then this for loop will take up. Now you can see that I am incrementing the range from 0 to 10 with an increment of 2. Therefore, it will go as 0, 2, 4, 8 and then 10 it will not print, because you are allowing it to go only up to 9. Therefore, it prints up to 8 only and comes out.

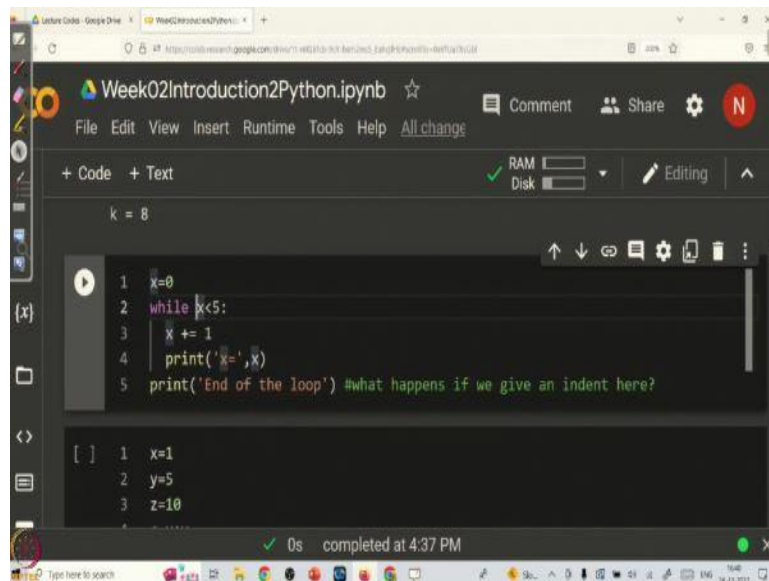
(Refer Slide Time: 12:05)



```
Week02Introduction2Python.ipynb
File Edit View Insert Runtime Tools Help All change
+ Code + Text
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
j = 3, j = 4, j = 5, j = 6, j = 7, j = 8, j = 9,
k = 0
k = 2
k = 4
k = 6
k = 8
0s completed at 4:37 PM
```

Let us run this code. The first for loop you have already seen. In the second for loop if you recall we asked the for loop to print one after the other that is how it went and in the third for loop we have printed $k = 0, 2, 4, 6$ and 8.

(Refer Slide Time: 12:21)



```
k = 8

1 x=0
2 while x<5:
3     x += 1
4     print('x=',x)
5 print('End of the loop') #what happens if we give an indent here?

[ ] 1 x=1
    2 y=5
    3 z=10

0s completed at 4:37 PM
```

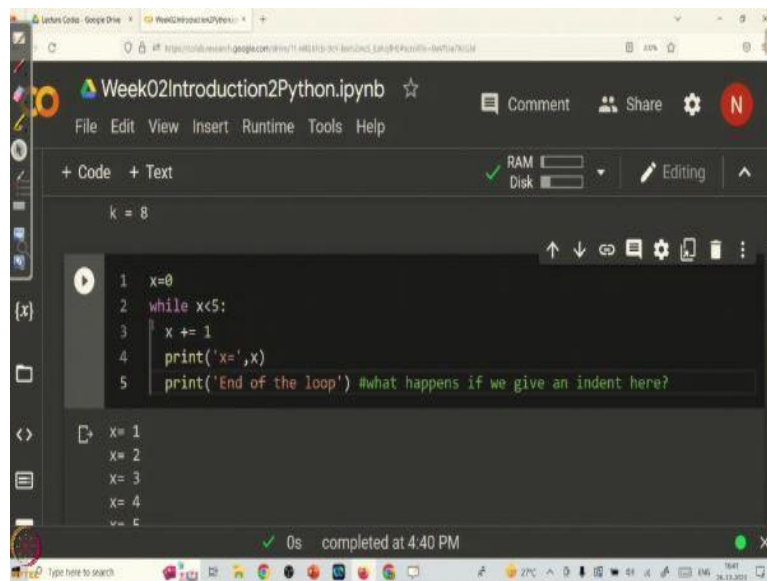
In the next program, we will see how to use while loop. Here, the format of the while loop is while and then the condition under which this loop has to run that is as long as this condition is satisfied the loop will keep on running. And then this colon is just like in the, for loop it is a format of the while loop as well as for loop also. It indicates that the line is ending there and then whatever is given as the indent here that will be considered as the part of the while loop.

Here, you can see that we have two lines, which are written with indent. Therefore, they are considered as the part of the while loop. And these two lines will be executed as long as the loop is going on and once the loop is over that is once if this condition is violated then the control comes out of the loop and prints this line. That is what the program does. **(Video Starts: 13:40)**

Let us run the program and see how the output looks like. You can see that first you have taken $x = 0$ and then you went into the loop first you are checking what is x value, x is surely less than 5, therefore it incremented x by 1, what it means? It is equivalent to saying $x = x + 1$. And then you are printing x , therefore the first print command will print the output as $x = 1$ and then it goes back to the while loop and checks the condition x is less than 5 because the value of x is 1.

Again, it comes and does this and gives $x = 2$ and it goes on up to $x = 5$, when it comes for 5 it prints and goes back to the while loop, it checks whether 5 is less than 5. No. Therefore, it will come out and then since this print command is not given with an indent. It will print not as the part of the while loop, but it will print once it comes out of the while loop. **(Video Ends: 14:54)**

(Refer Slide Time: 14:58)

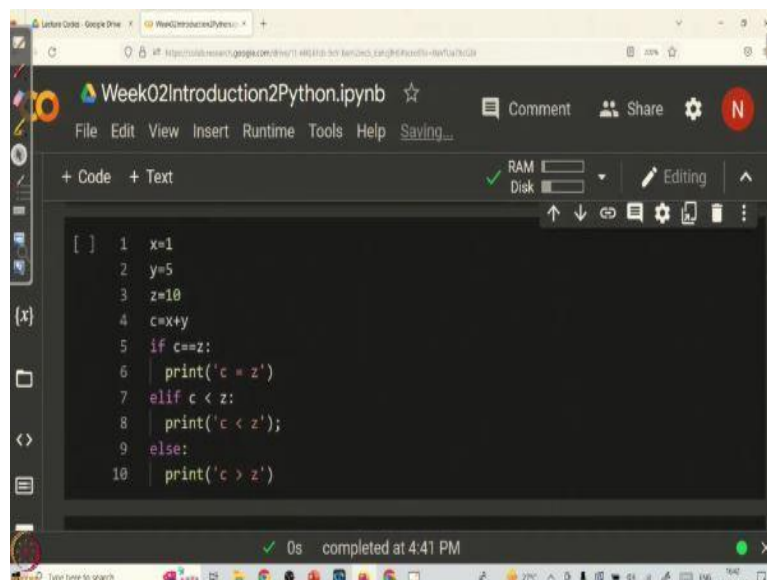


```
Week02Introduction2Python.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
RAM
Disk
Editing
k = 8
1 x=0
2 while x<5:
3     x += 1
4     print('x=',x)
5     print('End of the loop') #what happens if we give an indent here?
x= 1
x= 2
x= 3
x= 4
0s completed at 4:40 PM
```

Now, the question is what happens if I also give an indent to this? That is, if I just put a tab and push it inside and align with these lines, then this print will also be a part of this while loop.

(Video Starts: 15:11) Now if I run the code what happens every time it also prints that end of loop command, because it is a part of the while loop now, but my aim is not to keep it as a part of the while loop. So, let me remove that. **(Video ends: 15:27)** Let us go on with the next program.

(Refer Slide Time: 15:34)



```
Week02Introduction2Python.ipynb
File Edit View Insert Runtime Tools Help Saving...
+ Code + Text
RAM
Disk
Editing
1 x=1
2 y=5
3 z=10
4 c=x+y
5 if c==z:
6     print('c = z')
7 elif c < z:
8     print('c < z');
9 else:
10    print('c > z')
```

The next program is an illustration of the if command that is the conditional command if, else if and else. This is very easy for you to understand. Therefore, I leave it to you to understand this program, it is not very difficult.

(Refer Slide Time: 15:54)


```
1 A = [1, 2, 3] #this is called list in python
2 for i in range(4,11):
3     A.append(i) #extending the list size and assigning the value of i in it
4     print('A = ',A)
```

```
A = [1, 2, 3, 4]
A = [1, 2, 3, 4, 5]
A = [1, 2, 3, 4, 5, 6]
A = [1, 2, 3, 4, 5, 6, 7]
A = [1, 2, 3, 4, 5, 6, 7, 8]
A = [1, 2, 3, 4, 5, 6, 7, 8, 9]
A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

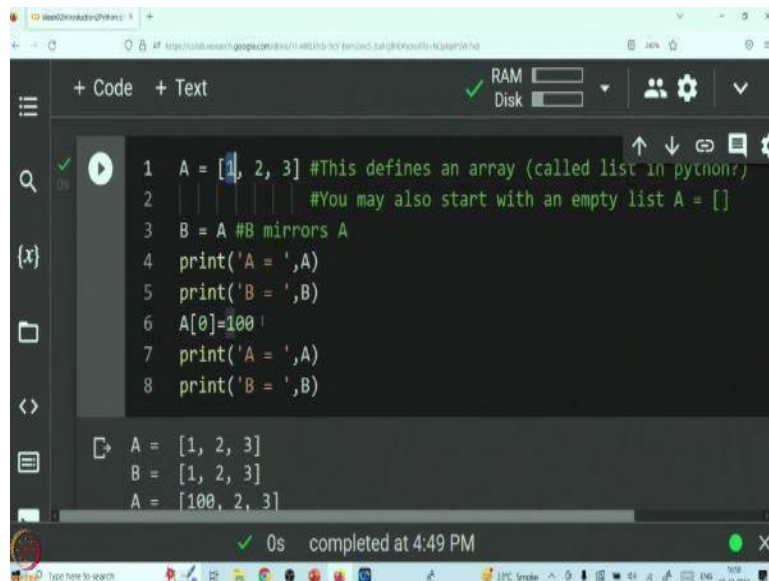
0s completed at 4:44 PM

The next program is an illustration of creating an array and extending the array length. Here we have a variable A in that we want to store a vector whose components are 1, 2 and 3. So, this can be done by creating an array. In Python arrays are called lists. And then what I want to do is I want to make this vector bigger every time when this for loop runs. For that what you have to do is A which is the variable name dot append that will create one more memory space for the variable A.

And it stores the value of i in that newly created variable. That is what it does. And every time after extending the array's length and storing the value of i into the new extended memory it prints the entire array A and then it does it for i = 4, 5, 6 up to 10. Let us run this code and see how the output looks like. You can see that when the control comes for the first time into the for loop the i value = 4, A already had 3 components.

Now, it has appended one more component in that it has stored the variable value i into it which is 4 now, that is why you see the variable output as 1, 2, 3 and 4 and then it goes back once again to the for loop. Now i = 5 when it comes to this line it again extends A by one more memory and puts the value of i which is now 5 into it and then prints it. That is why you can see you have 4 and then one more 5. Similarly, it goes on till i = 10 not 11. Again and again I am emphasizing, because this is little confusing it goes from 4 till 10.

(Refer Slide Time: 18:14)



```
1 A = [1, 2, 3] #This defines an array (called list in python)
2 #You may also start with an empty list A = []
3 B = A #B mirrors A
4 print('A = ',A)
5 print('B = ',B)
6 A[0]=100
7 print('A = ',A)
8 print('B = ',B)
```

A = [1, 2, 3]
B = [1, 2, 3]
A = [100, 2, 3]

0s completed at 4:49 PM

Well, this is how you create a one-dimensional array. We will now see the danger in equating an array with another variable name. Let us consider this program, where I am declaring a list or an array $A = [1, 2, 3]$ and then what I am doing is I am equating $B = A$. Normally, this should create a memory separately for the variable B and it should store the values of A into B and B should have the same list structure as A, but it should have its own memory.

But, in Python what happens is, B will not have its own memory, but it will be mirrored with A. And therefore, any changes you make in B will also be reflected in A and vice versa. Let us see this by first printing the values of A and B and then go to store some value in one of the components of either A or B and that will be automatically reflected in the other variable also. This is the danger of creating a variable B by equating it with a list. **(Video Starts: 19:51)**

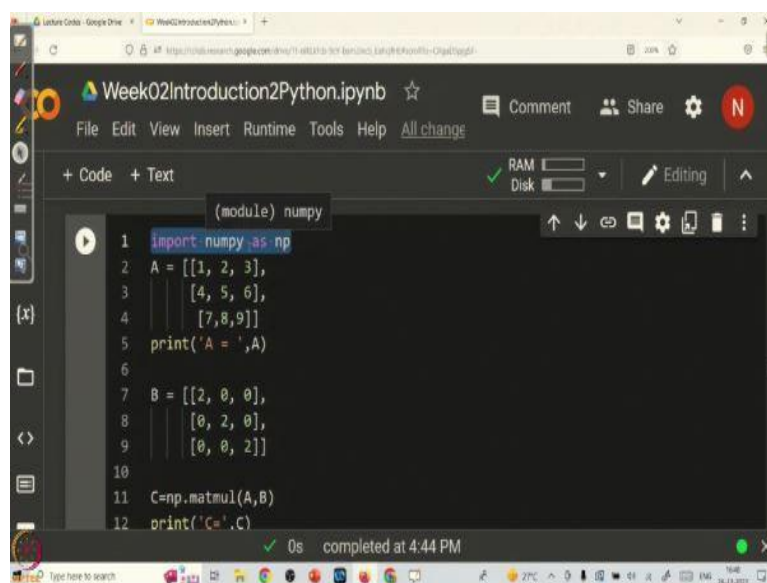
Let us run this program and see how the output comes. Well, you can see that $A = [1, 2, 3]$. And then since we have written $B = A$, we also got $B = [1, 2, 3]$. And finally I am changing the value of $A[0]$. That is the first component of A is changed at this line as 100. That is correctly incorporated in A now. You can see after changing that I am printing the value of A here and that is correctly taking care of the change happened in the first component of A, but I never touched B.

You see I have only created B and stored A into it, but I have never changed the value of B in any of the three components. But, since I have changed the first component of A from 1 to 100 it got reflected in the variable B also. This is the danger of equating a list to a new variable. On

the other hand, if you just put like this and see what happens now, you can see that only A got changed as we wanted B is never touched upon. **(Video Ends: 21:22)**

So, when you are defining a list and then you want to create another variable, which initializes what is there in A then just do not put `B = A`, that is very dangerous, but you put `B = A[:]` this is what is important when you are dealing with lists or otherwise called arrays. And now we have understood the one-dimensional arrays and their functionality. Let us go to see how to create and use two-dimensional arrays.

(Refer Slide Time: 22:03)



```
(module) numpy
1 import numpy as np
2 A = [[1, 2, 3],
3      [4, 5, 6],
4      [7, 8, 9]]
5 print('A = ',A)
6
7 B = [[2, 0, 0],
8      [0, 2, 0],
9      [0, 0, 2]]
10
11 C=np.matmul(A,B)
12 print('C=',C)
```

The screenshot shows a Jupyter Notebook interface with a dark theme. The code cell contains 12 lines of Python code. Line 1 imports NumPy as 'np'. Lines 2-4 define a 3x3 matrix A. Line 5 prints A. Line 6 is a blank line. Lines 7-9 define a 3x3 matrix B. Line 10 is a blank line. Line 11 uses np.matmul to multiply A and B, storing the result in C. Line 12 prints C. The notebook shows the code is completed at 4:44 PM.

A two-dimensional array can be created like this. Now a can be viewed as a matrix, whose elements are `[[1, 2, 3], [4, 5, 6], [7,8,9]]`. In order to emphasize that you do not need to give a space between comma and the number. So, I have written the last line without the space. So, that does not really matter. So, what I am doing here is I am basically defining two matrices A and B and then I want to multiply these two matrices.

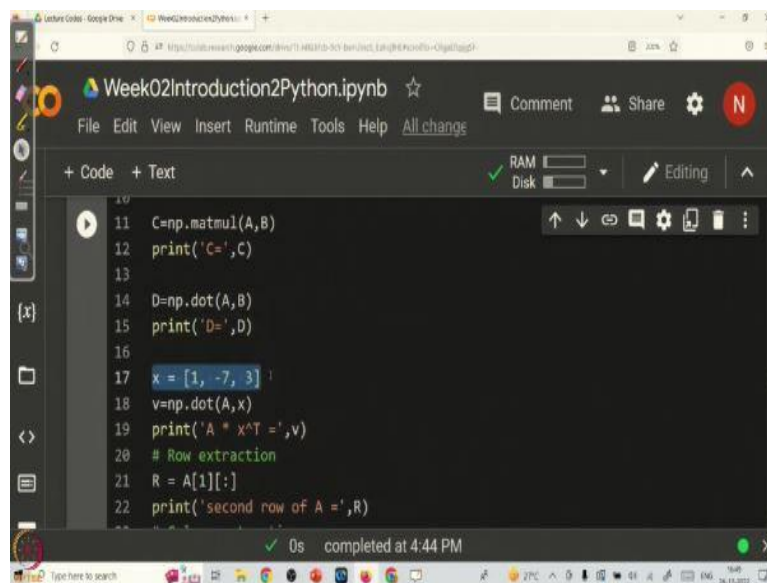
You do not have an inbuilt command in Python to multiply two matrices. What you can do is you can create two nested for loops and you can directly make the multiplication formula and get the product of two matrices or what you can do is there are some inbuilt libraries in Python; say for matrix multiplication you can find commands in a inbuilt library called numpy.

So, how will you bring that numpy into your program? You have to first import it, this is what we are doing in the first line. In fact you can do it anywhere before you use any command of that library or generally it is a good practice to define all the libraries that you are going to use

in your program at the beginning of the program. So, I am just defining it at the beginning of the program.

You can also define it at this level, because this is the first time you are using a numpy command. So, you can also either put it here or you can put it at the beginning of the program. So, what it does is, it imports all the commands, which are written and kept in this library called numpy.

(Refer Slide Time: 24:18)



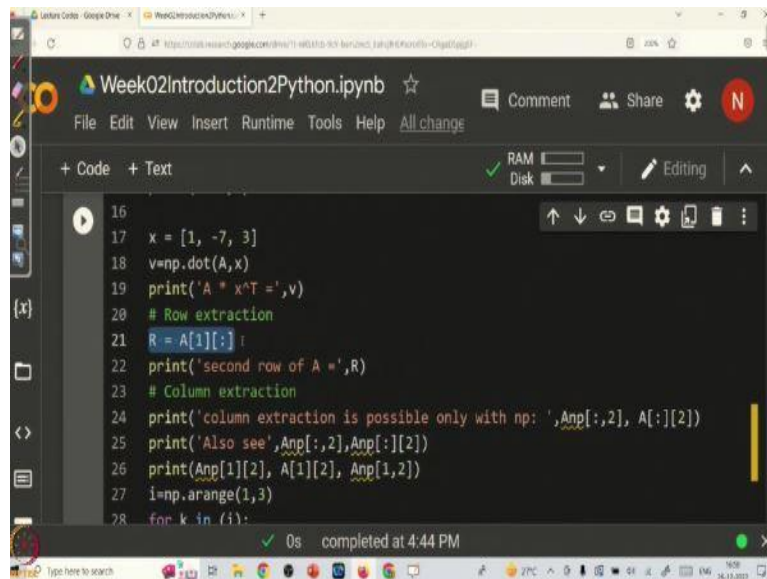
```
11 C=np.matmul(A,B)
12 print('C=',C)
13
14 D=np.dot(A,B)
15 print('D=',D)
16
17 x = [1, -7, 3]
18 v=np.dot(A,x)
19 print('A * x^T =',v)
20 # Row extraction
21 R = A[1][:]
22 print('second row of A =',R)
```

Then whenever you call any of its commands, say for instance `matmul`, will do the matrix multiplication of given two matrices A and B, but to execute it you have to say that this command is sitting in the library called numpy. So, you have to type `numpy dot matmul`. If you do not want to type this full name then you have to import it as something. So, what I am choosing is I want to use instead of `numpy np` I want to use.

You can give any string here and then you can use that string to call the library I am using np therefore I have to put `np.matmul` this is matrix multiplication of A comma B and that I am storing in the variable C. You can also use `dot` for multiplying a matrix with a vector, which I am doing here, you can see that I am multiplying the matrix A with x. Although, this x is looking like a row vector in the dot command it will be automatically taken as a column vector.

And it gives you the correct result of A into x. So, therefore when you print this you will see the value A into x transpose here that is why I have just printed like this.

(Refer Slide Time: 25:48)

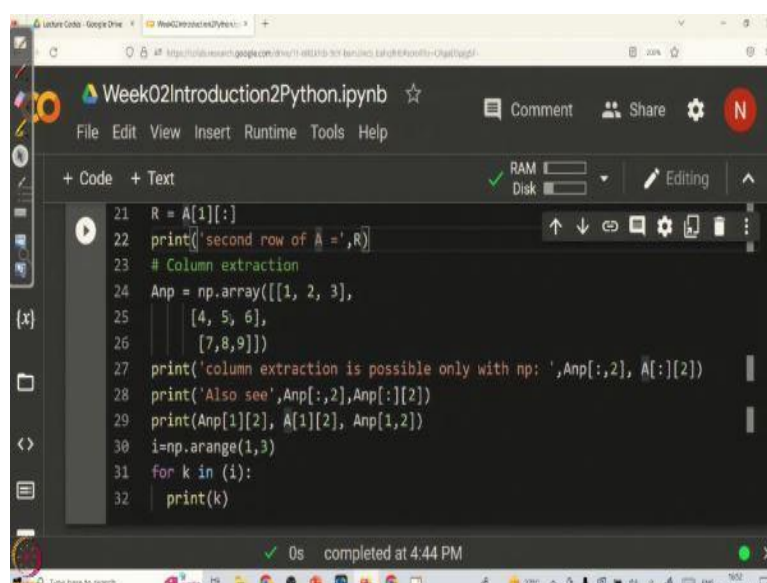


```
16
17 x = [1, -7, 3]
18 v=np.dot(A,x)
19 print('A * x^T =',v)
20 # Row extraction
21 R = A[1][:]
22 print('second row of A =',R)
23 # Column extraction
24 print('column extraction is possible only with np: ',Anp[:,2], A[:,2])
25 print(' Also see ',Anp[:,2],Anp[:,2])
26 print(Anp[1][2], A[1][2], Anp[1,2])
27 i=np.arange(1,3)
28 for k in (i):
```

And in fact you can extract one particular row of the matrix A by giving this command. Remember in Python the index always starts with 0, therefore when I give the index as 1 it is basically the second row, but not the first row. Because, first row is indexed as 0 in Python, it is something like C program the index starts always with 0. You have to remember that that is why I have written here print second row of A but I have given 1 here why because 0 corresponds to the first row of A, 1 corresponds to the second row of A.

Now, if I am interested in extracting a column can I put A colon 1? Well that is not possible, because colon 1 is not going to extract a column. So, column has to be done in a rather different way.

(Refer Slide Time: 27:00)



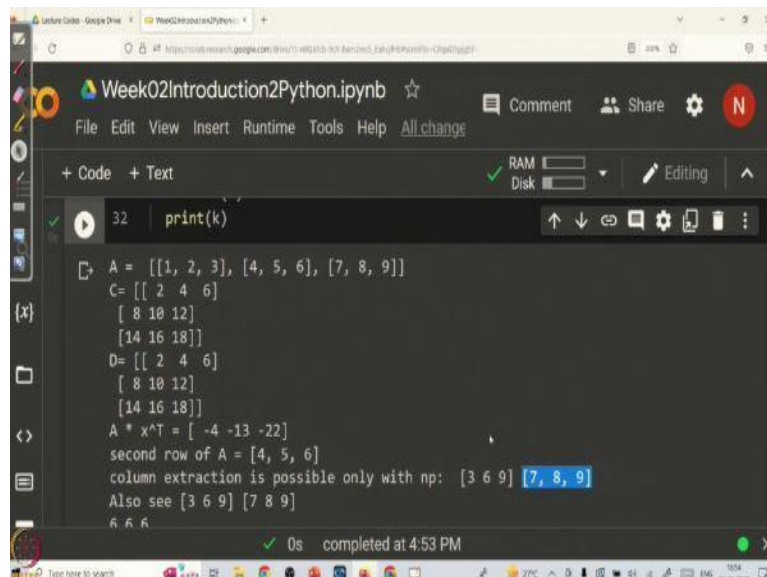
```
21 R = A[1][:]
22 print('second row of A =',R)
23 # Column extraction
24 Anp = np.array([[1, 2, 3],
25               [4, 5, 6],
26               [7,8,9]])
27 print('column extraction is possible only with np: ',Anp[:,2], A[:,2])
28 print(' Also see ',Anp[:,2],Anp[:,2])
29 print(Anp[1][2], A[1][2], Anp[1,2])
30 i=np.arange(1,3)
31 for k in (i):
32     print(k)
```

In order to extract a column you cannot define your matrix A in the normal list form you have to define it as np dot array of this. So, it this is a special command, which is written and kept in the library numpy. So, you have to use that command to declare the matrix A just to have a different notation. Let me call this as Anp. You can see that the matrix A and Anp are mathematically the same.

They are the same matrices, but A is defined using the usual list command of the Python, whereas Anp is created using the command, which is kept in the numpy. Why we are doing it? With this now you can extract a column otherwise you can only extract rows. You cannot extract columns. Let us print Anp and now if you want to say extract the third column of the matrix A.

Remember, I am putting 2 here, but actually it extracts the third column, because again I am emphasizing Python will run from 0 to n- 1. Therefore 2 will correspond to the third column and then A of colon comma 2 this will not extract the column. You can run this program and see what is the output of this.

(Refer Slide Time: 28:51)

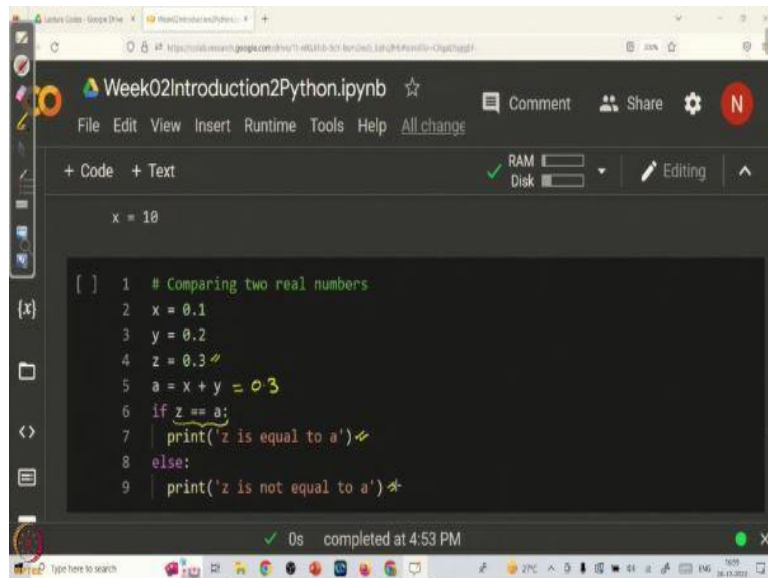


```
Week02Introduction2Python.ipynb
File Edit View Insert Runtime Tools Help All change
+ Code + Text
RAM
Disk
Editing
32 print(k)
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
C= [[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
D= [[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
A * x^T = [ -4 -13 -22]
second row of A = [4, 5, 6]
column extraction is possible only with np: [3 6 9] [7, 8, 9]
Also see [3 6 9] [7 8 9]
6 6 6
0s completed at 4:53 PM
```

You can see that the column extract is possible only with np. You can see you get A is given like this and we are asking it to extract the third column. The third column is correctly extracted as 3, 6 and 9 with Anp. Remember this first one is Anp and whereas the second one is not extracting the third column, but it still extracted the third row only. So, that is the main drawback of defining a two-dimensional array directly with the inbuilt list form you have to define a matrix always with np dot array command.

We will learn more about handling matrices, when we are doing the linear systems path. Let us now quickly go into the next topic of doing some programs with rounding errors. Let us see this interesting code.

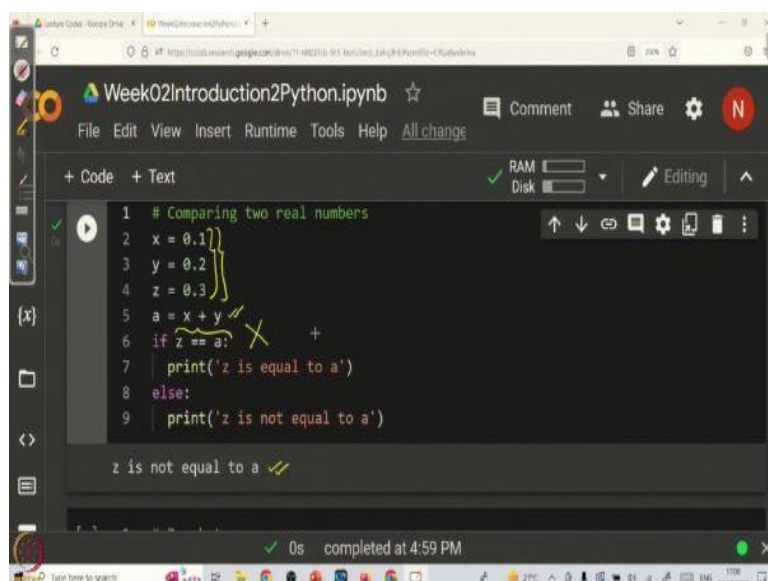
(Refer Slide Time: 30:00)



```
Week02Introduction2Python.ipynb
File Edit View Insert Runtime Tools Help All change
+ Code + Text RAM Disk Editing
x = 10
[ ] 1 # Comparing two real numbers
    2 x = 0.1
    3 y = 0.2
    4 z = 0.3
    5 a = x + y = 0.3
    6 if z == a:
    7     print('z is equal to a')
    8 else:
    9     print('z is not equal to a')
```

The code says that I am storing the value 0.1 in x, y = 0.2 and z = 0.3. Then I am adding a = x + y what will be the value of a? Mathematically, a will be equal to 0.3. Then what I am doing is I am comparing z with a. If z = a it should give me yes z = a that is what the print command should say. Otherwise, it should say z is not equal to a. Obviously we will expect that the output of this program should be z = a. Let us see what is the output of this program?

(Refer Slide Time: 30:53)



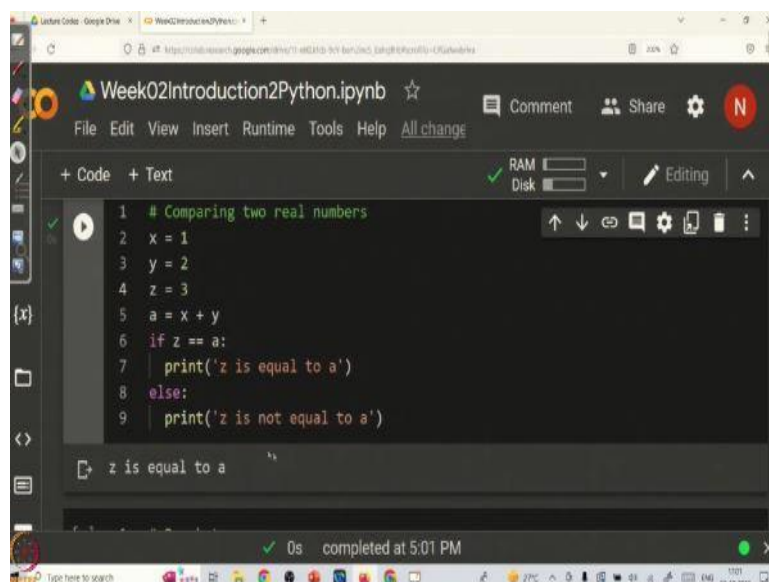
```
Week02Introduction2Python.ipynb
File Edit View Insert Runtime Tools Help All change
+ Code + Text RAM Disk Editing
1 # Comparing two real numbers
2 x = 0.1
3 y = 0.2
4 z = 0.3
5 a = x + y
6 if z == a:
7     print('z is equal to a')
8 else:
9     print('z is not equal to a')
```

z is not equal to a

The program is executed and surprisingly you can see that the code has written z is not equal to a. Why this is happening? Well, this is not the problem with Python. You write the same code in any language that you know you can write it on Matlab or any language you will see that the output is going to be z is not equal to a. This is because when your processor does this computation it does not do exactly, even though the numbers are very, very small numbers and they are nice looking numbers also.

Therefore, comparing such things when you are using non integer values are going to be very dangerous never do like this. You should always keep a range of numbers and compare them only within that range. You cannot do it with absolute comparison. For instance, if you take x, y and z as 1, 2, 3 instead of 0.1, 0.2 and 0.3 then the output will be as you expect let us see what is the output of this program. When I take x = 1, y = 2 and z = 3.

(Refer Slide Time: 32:24)

A screenshot of a Jupyter Notebook interface. The notebook is titled "Week02Introduction2Python.ipynb". The code cell contains the following Python code:

```
1 # Comparing two real numbers
2 x = 1
3 y = 2
4 z = 3
5 a = x + y
6 if z == a:
7     print('z is equal to a')
8 else:
9     print('z is not equal to a')
```

The output cell shows the result: "z is equal to a". The status bar at the bottom indicates "0s completed at 5:01 PM".

Then it will print what you actually expect. Therefore, this simple and interesting program says that computer even dealing with small numbers it has its very serious limitations on the rounding errors.

(Refer Slide Time: 32:42)

```
[ ] 1 # Bracket sum
2 x = (0.8 + 0.1) + 0.3 ✓
3 y = 0.8 + (0.1 + 0.3) ✓
4 c = 10**16 ✓, finite > 0
5 z = (1 - x/y)*c ✓, x/y ≈ 1
6 print('z = ',z) ✓, z=0

[ ] 1 x = 1.e15
2 y = -1.e15
3 z = 1
```

Let us see this next interesting problem. I am taking $x = 0.8 + 0.1 + 0.3$. I am only bracketing the sum in a different way, otherwise you can observe that x and y are going to be the same and then I want to find x/y . So, I want to find x/y . Since, $x = y$ this will be mathematically equal to 1. But, since you are doing the arithmetic with real numbers they both may not have exactly equal value.

Therefore, this may not be exactly 1 but only approximately equal to 1. Now, what I am doing? I am subtracting 1 from x/y mathematically this should give me the value 0. And then just to have a catchy output I am multiplying it with a big number. Now, whatever may be this big number it is finally a finite number. Therefore 0 into a finite number should be equal to 0. So, we expect this print command to print $z = 0$. Let us see what is the output of this program?

(Refer Slide Time: 34:13)

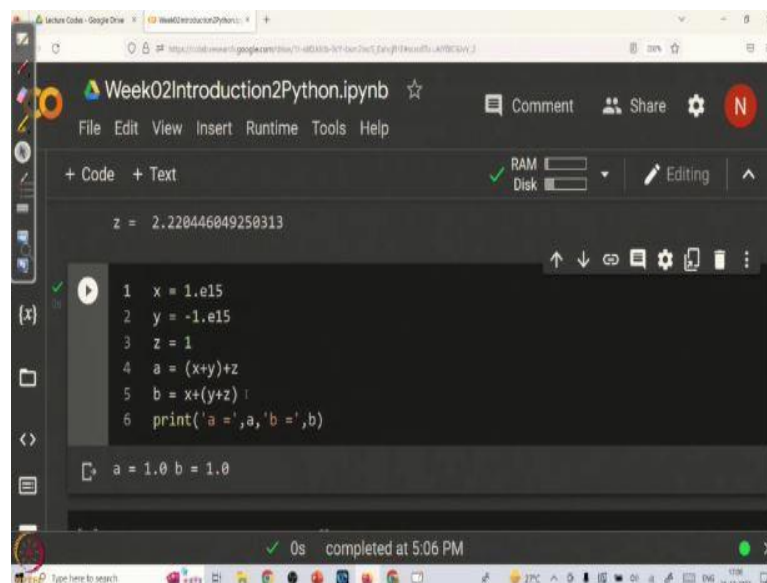
```
[ ] 1 # Bracket sum
2 x = (0.8 + 0.1) + 0.3
3 y = 0.8 + (0.1 + 0.3)
4 c = 10**16
5 z = (1 - x/y)*c
6 print('z = ',z)

z = 2.220446049250313

[ ] 1 x = 1.e15
2 y = -1.e15
3 z = 1
```

You can see that the output is significantly large number. You see we expect the output to be equal to 0 but computer made a huge error in this simple computation and this shows how dangerous it is for us to deal with computation especially, when we do such computations blindly we may very likely land up with some disastrous answers. This also gives us a strong motivation to do a deeper analysis on numerical methods that we do before going into the implementation of those methods. We have to be very clear about how these methods are going to work and how we implement this method on a computer.

(Refer Slide Time: 35:05)



The screenshot shows a Jupyter Notebook interface. At the top, the title is "Week02Introduction2Python.ipynb". Below the title bar, there are tabs for "Code" and "Text". The code cell contains the following Python code:

```
z = 2.220446049250313

1 x = 1.e15
2 y = -1.e15
3 z = 1
4 a = (x+y)+z
5 b = x+(y+z)
6 print('a =', a, 'b =', b)
```

The output of the code cell is:

```
a = 1.0 b = 1.0
```

At the bottom of the code cell, there is a status bar that says "0s completed at 5:06 PM".

The next is let us take this another interesting program. Let us execute this program and see what is the output of this program? Well, as expected you can see that a and b should be mathematically equal and that is what is also shown in this program. Now, instead of 10 to the power of 15 let me make it as 10 to the power of 16 here and similarly here minus 10 to the power of 16 and run this program now you can see that a is still equal to 1, but b is not equal to 1.

What happened is 10 to the power of 16 is something, which has gone beyond the memory of the computer and that made this huge error between a and b.

(Refer Slide Time: 36:00)

```

1  ### e > 1023 => overflow
2  e = 1021
3  x=2.**e
4  for i in range(0,3):
5      e += 1
6      x=2.**e
7      #x*=2
8      y = 1/x
9      z = x/x
10     print('e = ',e,'x = ',x,'y = ',y,'z = ',z)

```

0s completed at 5:06 PM

Let us try to understand this more closely. Let us take this program. I have taken $e = 1021$ and we have $x = 2$ to the power of e . Now, I am going to increment my exponent e and every time I am going to save $x = 2$ to the power of e . This 2 stars means, it is rise to the power. It is equivalent to saying that this is nothing but 2 to the power of e mathematically. So, therefore you have a loop running from 0 to 2 every time I am incrementing e by 1 and then rising 2 to the power of e and saving that value into x and then I am finding $y = 1$ by x , $z = x$ by x and then printing all this. Let us see what is the output of this program.

(Refer Slide Time: 37:04)

```

8  y = 1/x
9  z = x/x
10 print('e = ',e,'x = ',x,'y = ',y,'z = ',z)

```

```

e = 1022 x = 4.49423283715579e+307 y = 2.2250738585072014e-308 z = 1.0
e = 1023 x = 8.98846567431158e+307 y = 1.1125369292536007e-308 z = 1.0

```

```

OverflowError                                Traceback (most recent call last)
<ipython-input-14-e6248ab61db4> in <module>
      4 for i in range(0,3):
      5     e += 1
----> 6     x=2.**e
      7     #x*=2
      8     y = 1/x

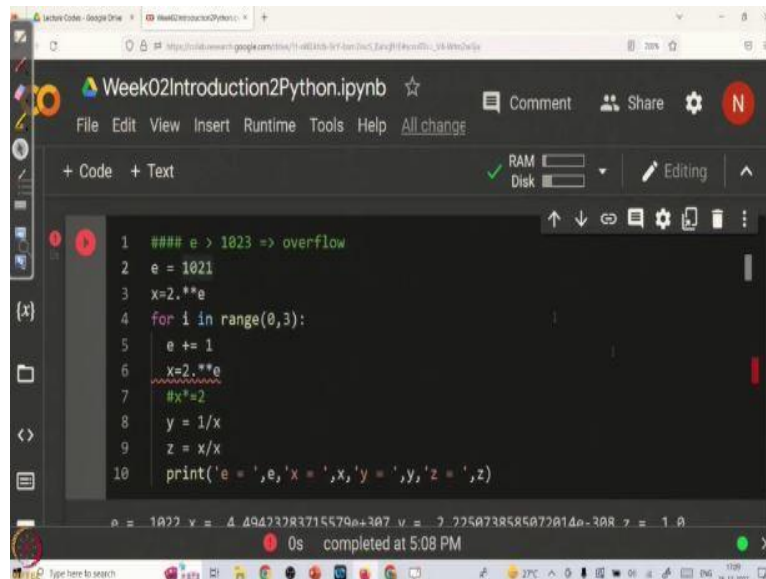
```

0s completed at 5:08 PM

Will the program give an error that is because you are explicitly rising 2 to the power of e , where this e is going beyond the memory capacity. In the theory class, we have seen that this is what we call as overflow of memory. When you are rising 2 to the power of e and the e is

going beyond the maximum memory capacity of the computer then Python is automatically recognizing it and giving an error command.

(Refer Slide Time: 37:44)

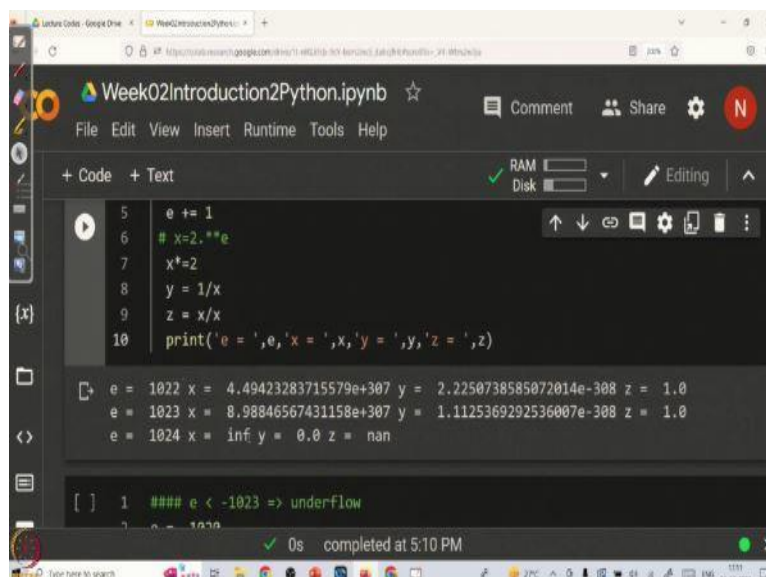


```
1 ##### e > 1023 => overflow
2 e = 1021
3 x=2.**e
4 for i in range(0,3):
5     e += 1
6     x=2.**e
7     #x*=2
8     y = 1/x
9     z = x/x
10    print('e = ',e,'x = ',x,'y = ',y,'z = ',z)
```

Completed at 5:08 PM

Let us fool the Python by not putting this command, but define the same expression in a different way that is now what I am going to do is, I am not going to increment e and then rise 2 to the power of e. What I am going to do is, I already have $x = 2$ to the power of e, where e is 1021. Now, what I will do is every time I will multiply x by e. This is nothing but $x = x$ into e this one. Now, let us see what happens?

(Refer Slide Time: 38:25)



```
5 e += 1
6 # x=2.**e
7 x*=2
8 y = 1/x
9 z = x/x
10 print('e = ',e,'x = ',x,'y = ',y,'z = ',z)
```

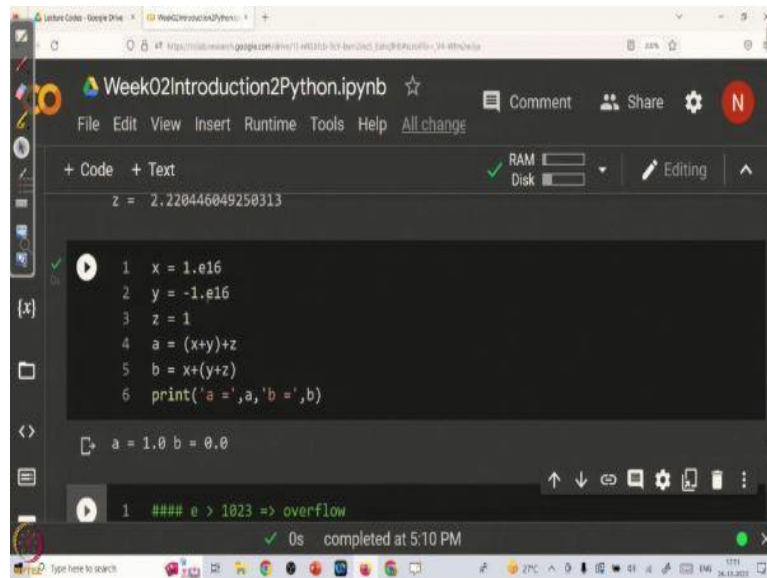
Completed at 5:10 PM

Now, Python will not recognize it as an error, it runs, because you are not defining it in a nice way like 2 to the power of e. You are not doing it, with that it is able to recognize the overflow of memory, but with this it is not able to recognize the overflow of memory. It went on and

computed it got x as infinity if you recall if the overflow of memory happens. Then the computer processor will treat it as infinity then $y = 1$ by infinity, which has printed as 0.

And z is equal to infinity by infinity, which is called as not a number nan, it has printed it as nan. From here you can see that the overflow of memory happens when the exponent is 1024. This is 2 to the power of 1024 is the limit of the memory of the computer.

(Refer Slide Time: 39:25)



```
Week02Introduction2Python.ipynb
File Edit View Insert Runtime Tools Help All change
+ Code + Text
RAM
Disk
Editing
z = 2.220446049250313
1 x = 1.e16
2 y = -1.e16
3 z = 1
4 a = (x+y)+z
5 b = x+(y+z)
6 print('a =', a, 'b =', b)
a = 1.0 b = 0.0
1 ### e > 1023 => overflow
0s completed at 5:10 PM
```

In terms of 10 it is 10 to the power of 16. In terms of 2 it is 2 to the power of 1024. So, that is what is happening here. So, there was an overflow of memory. Similarly, you can also play around with the underflow of memory; I will leave it to you to play around with this code. You just write this code and test it on your computer or on any programming language. This will happen not only with Python not only on Colab, it will happen with any programming language on any operating system. As long as you are using a 64 bit processor, this kind of overflow and underflow will always happen.

(Refer Slide Time: 40:10)

The screenshot shows a Jupyter Notebook titled "Week02Introduction2Python.ipynb". The code cell contains the following Python code:

```

6 x = 2**e
7 #x /= 2
8 print('e = ',e,'x = ',x,'1/x = ',1/x)

```

The output cell shows the result of the code execution:

```

1 ### Machine epsilon
2 xu=2**53
3 xo=2**53
4 y = 100 * xo * (xu+1-1);
5 print('y = ',y)

```

The output of the code is `y = 0.0`. Handwritten annotations in yellow and green explain the machine epsilon. It notes that the mantissa has 52 bits, and the machine epsilon is 2^{-52} , which is approximately 2^{-53} . The calculation $y = 100 * x_0 * (x_u + 1 - 1)$ is shown to result in $y = 100 * 0 = 0$ due to the machine epsilon limitation.

Finally, we will try to understand what is machine epsilon? Python always deals with it is variable by default in the double precision; it means it gives 52 bits in the mantissa. Therefore, in one of the problems that we have seen in the last tutorial class, if it is 52 bits it means 0 to 51 it goes. Therefore, from the 53rd digit onwards your processor will make a rounding approximation.

It takes up to 52 digits and then from the 53rd digit onwards it makes a rounding approximation. In the last class, we have seen that if such a thing happens then 2 to the power of minus n is the machine epsilon. So, that is what is given here. I am taking $x_u = 2$ to the power of -53 and x_o as 2 to the power of 53. Now, what I am doing is I am computing y as 100 into x_o that is a big number into $x_u + 1 - 1$.

Now you see this will be understood as 1 only in the computer. And therefore you will see that this is nothing but $1 - 1$, which is equal to 0. On the other hand, if this is not there then this and this will get canceled. You are supposed to get the value 100 as the result, but you will get the answer as 0 for this, because of this machine epsilon calculation. Let us see the output of this program, you can see that $y = 0$, whereas it is supposed to give us $y = 100$.

These are the limitations of the computer. This is not only for Python or Colab it is there for any program, because this limitation is coming from the processor of your computer not from the program path. So, these limitations have to be carefully understood while you are working with the scientific computation. Otherwise, you may be doing entirely wrong computation although your method mathematically may be correct.

Therefore, you have to first carefully understand all the limitations of the computer. And also you should do the arithmetic error analysis, of course also you have to do the mathematical error analysis coming from the method and then arithmetic error analysis has to be done. Then you go for the programming of your method. This is the correct approach to numerical analysis. With this let us finish this class. Thank you for your attention.