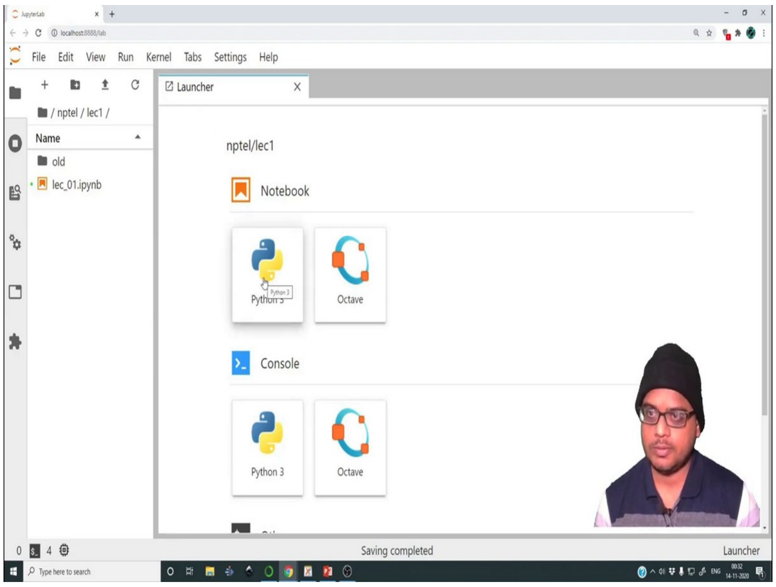
# Tools in Scientific Computing

**Prof. Aditya Bandopadhyay**
**Department of Mechanical Engineering**
**Indian Institute of Technology, Kharagpur**

## Lecture – 02
## Loops and Conditionals Implementation of bubble sort

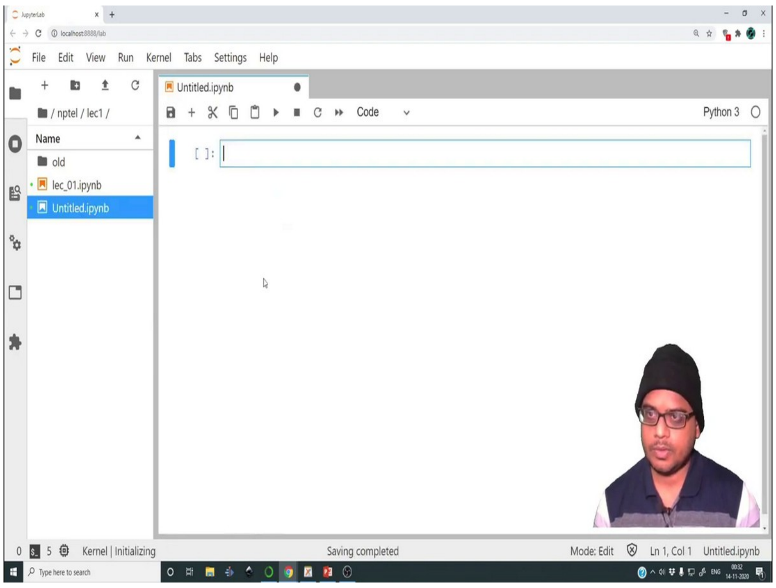
Hello everyone, this is the second lecture in Tools in Scientific Computing. In this lecture we are going to look at conditionals and loops.

(Refer Slide Time: 00:39)

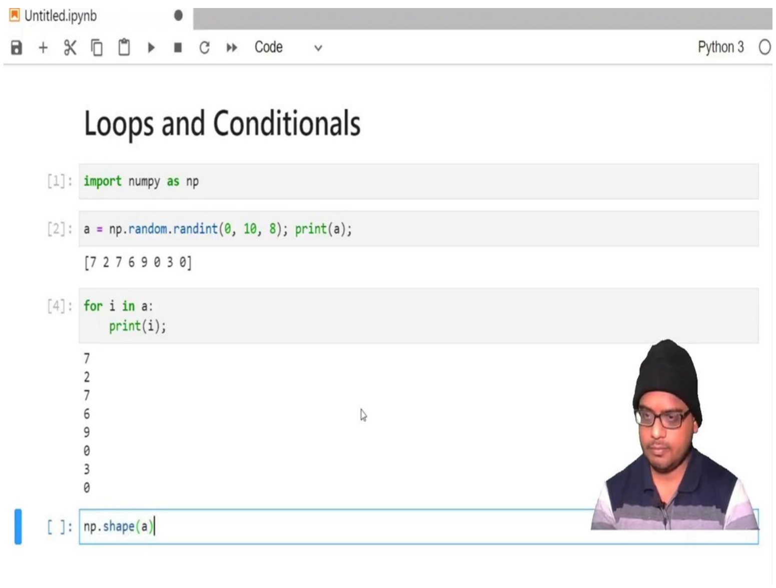


Upon opening the anaconda navigator we launch JupyterLab and we end up on this screen. So, let me create a new notebook I am using the python 3 kernel ok.

(Refer Slide Time: 00:50)



(Refer Slide Time: 00:57)

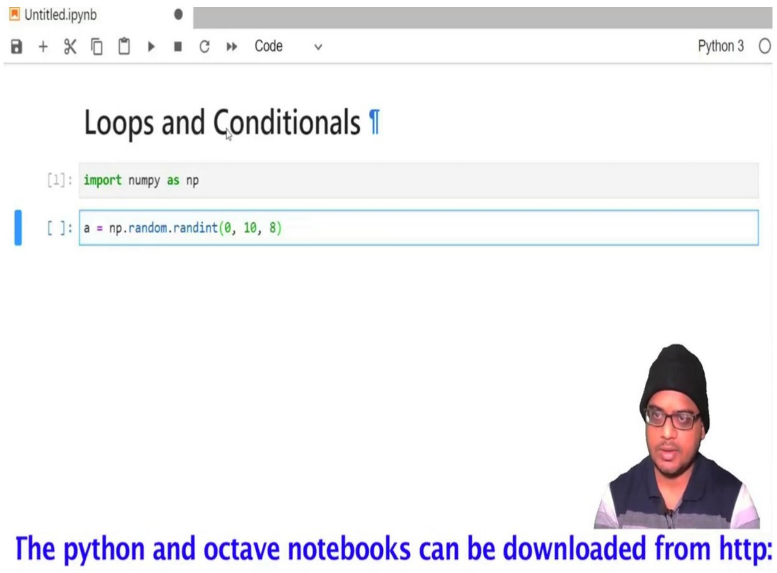


## Loops and Conditionals

```
[1]: import numpy as np

[2]: a = np.random.randint(0, 10, 8); print(a);
     [7 2 7 6 9 0 3 0]

[4]: for i in a:
         print(i);
     7
     2
     7
     6
     9
     0
     3
     0

[ ]: np.shape(a)
```

So, let us consider the creation of an array in order to demonstrate some of these ideas.

(Refer Slide Time: 01:03)



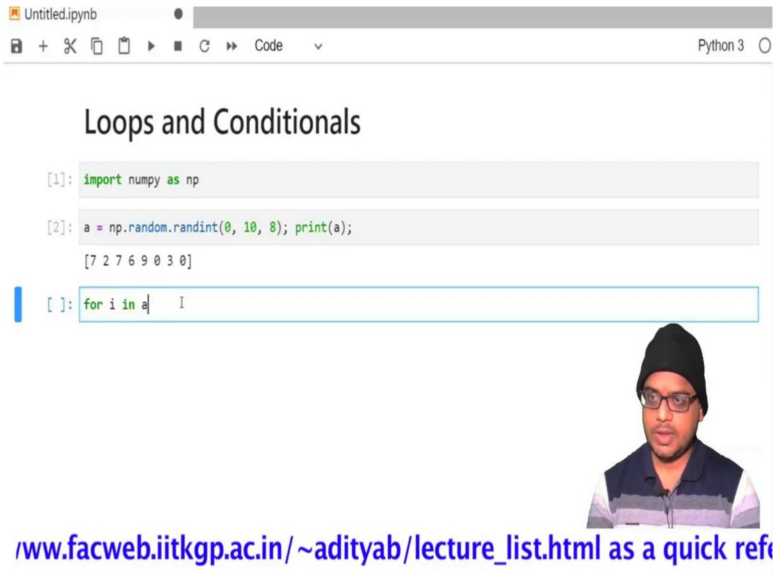
Loops and Conditionals ¶

```
[1]: import numpy as np
[ ]: a = np.random.randint(0, 10, 8)
```

The python and octave notebooks can be downloaded from http:

So, let me import numpy. So, let me create a random array. So, let me create a=np.random.randint(). So, this is a function randint it creates random integers between the given the specified range. So, in this case the range is 0 to 10; let me create a 1 dimensional array. So, let me have 8 elements in the array.

(Refer Slide Time: 01:35)



Loops and Conditionals

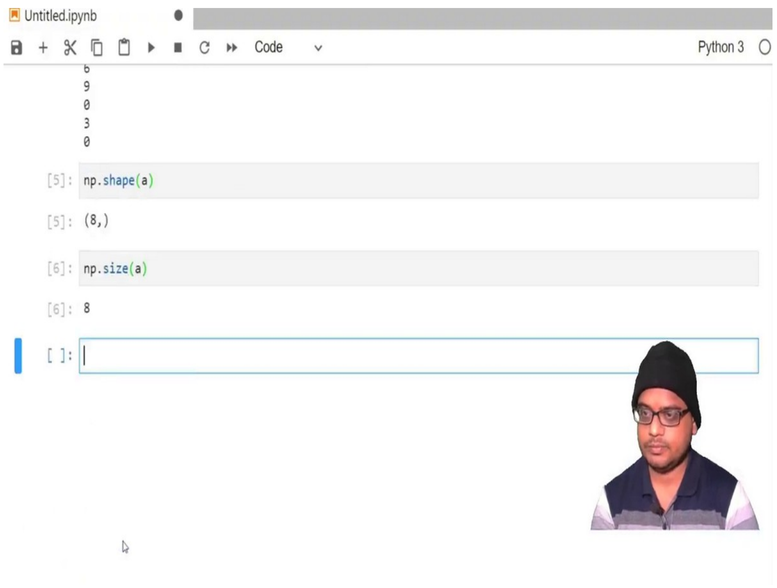
```
[1]: import numpy as np
[2]: a = np.random.randint(0, 10, 8); print(a);
     [7 2 7 6 9 0 3 0]
[ ]: for i in a
```

/ww.facweb.iitkgp.ac.in/~adityab/lecture_list.html as a quick refe

So, let me print out the value of a as well ok. So, we have a random array a and it has 8 elements. So, suppose I want to loop over all the elements of the array a and print them one by one. So, how do we do that?

So, we use a loop that is an iteration. So, we do for i in a so we we can simply do this. We can print i ok. So, this prints out all the values of the array a ok. This is a very easy way of printing out all the values. Let me query the length of the array a. So, I will write. So, let me print out this; so np.shape(a).

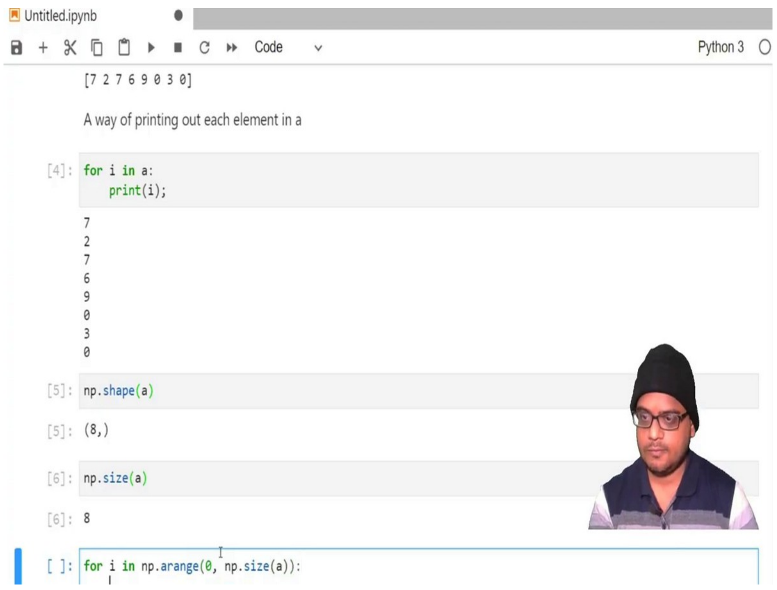
(Refer Slide Time: 02:50)



```
6
9
0
3
0
[5]: np.shape(a)
[5]: (8,)
[6]: np.size(a)
[6]: 8
[ ]: |
```

So, it says it is 8 rows and it has I mean it has 8 elements. So, let me print out the np.size(a). The size of a is 8 so this says the total number of elements that a has. So, what I can do is; I can instead of looping over all the elements in a. So, it means for I which belongs in a ok.

(Refer Slide Time: 03:30)



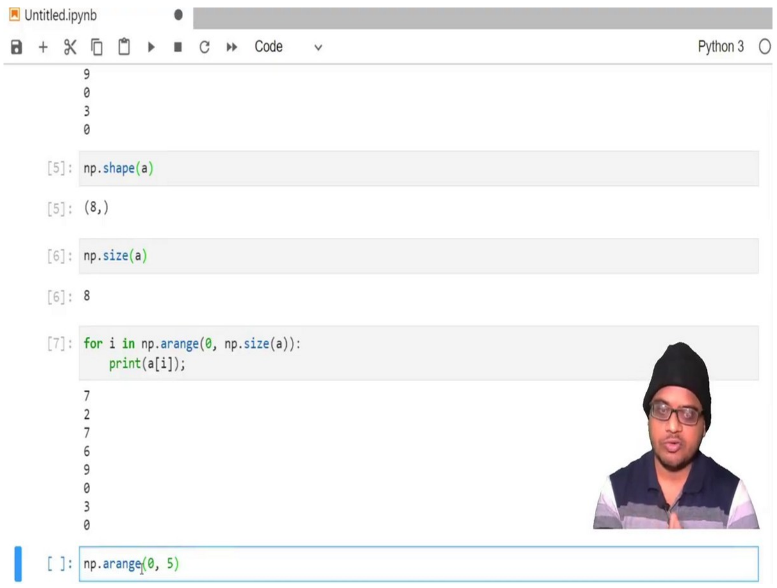
```
[7 2 7 6 9 0 3 0]
```

A way of printing out each element in a

```
[4]: for i in a:
         print(i);
7
2
7
6
9
0
3
0
[5]: np.shape(a)
[5]: (8,)
[6]: np.size(a)
[6]: 8
[ ]: for i in np.arange(0, np.size(a)):
```

So, this is so instead of looping it like this we could have alternately looped it like this for i in np.arange(0,np.size(a)).

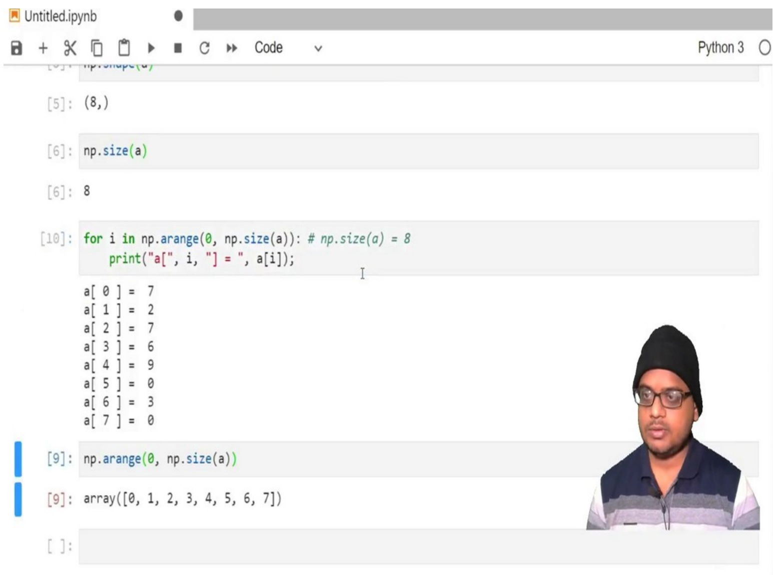
(Refer Slide Time: 04:00)



```
         9
         0
         3
         0

[5]:     np.shape(a)

[5]:     (8,)

[6]:     np.size(a)

[6]:     8

[7]:     for i in np.arange(0, np.size(a)):
             print(a[i]);

         7
         2
         7
         6
         9
         0
         3
         0

[ ]:     np.arange(0, 5)
```

Then we print a[i] ok. So, we have obtained all the elements in the array a; what is this np.arange()? So, let me show you. So, np.arange() is a way of obtaining an integer range from an initial number say suppose the number is 0 and the number of elements. So, the number of elements suppose I keep 5; so it should output 0, 1, 2, 3, 4.

(Refer Slide Time: 04:39)



```
[5]:     (8,)

[6]:     np.size(a)

[6]:     8

[10]:    for i in np.arange(0, np.size(a)): # np.size(a) = 8
             print("a[", i, "] = ", a[i]);

         a[ 0 ] =  7
         a[ 1 ] =  2
         a[ 2 ] =  7
         a[ 3 ] =  6
         a[ 4 ] =  9
         a[ 5 ] =  0
         a[ 6 ] =  3
         a[ 7 ] =  0

[9]:     np.arange(0, np.size(a))

[9]:     array([0, 1, 2, 3, 4, 5, 6, 7])

[ ]:
```

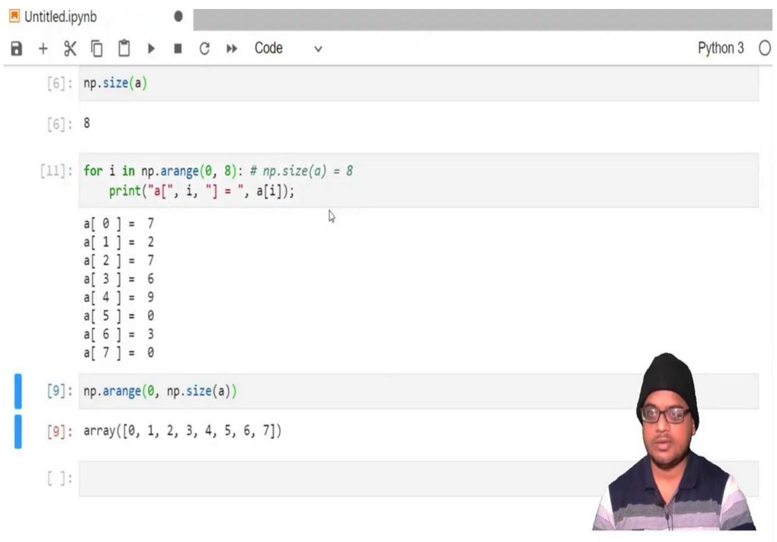
So, let me see ok; so it does output 0, 1, 2, 3, 4. So, the first argument to the function is the opening number and the last argument to this function the number of elements. So, let us dissect what we did over here. So, np.size(a) was equal to 8. So, for i in np.arange(0,np.size(a)); so let me actually print that out. So, it gives 0, 1, 2, 3, 4 all the way to 7.

Then i; so we loop using an iterator i and we print a[i]. So, in the first loop i takes the value 0 because for i in this particular array it will take the value 0. So, it will first print a 0, then it will print a 1, then it will print a 2, it will print a 3 and so on. In fact, let me prettify this print statement a bit better. So, here I will write print("a[",i,"]=", a[i]).

So, this is literal it will print out a opening bracket then it will print out the value of i then it will literally print out the closing bracket and the equal to sign then it will print out a[i] ok so here we have it.

So, this is how we can literally print all the elements of an array. So, this is an example. So, I have shown you 2 examples or 3 examples of printing out all the elements of an array. This is quite handy when you write codes of finite differences because you want to loop over various elements on the grid.

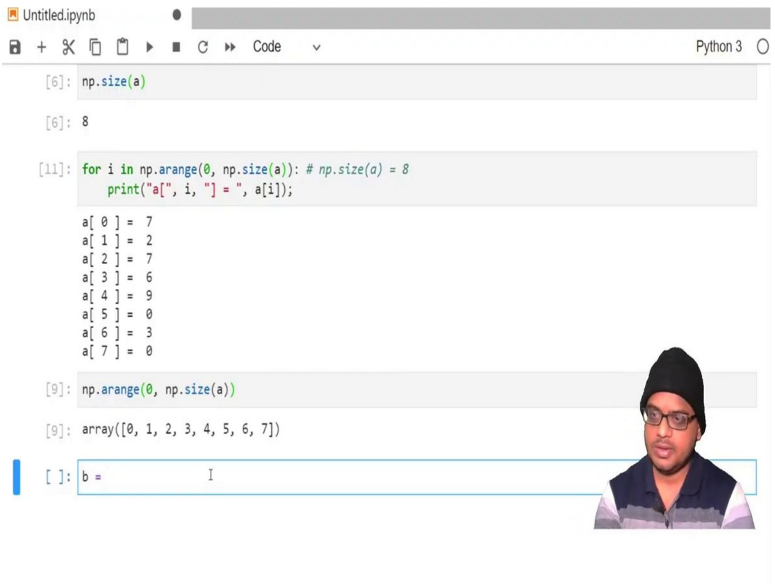
(Refer Slide Time: 06:51)



```
[6]: np.size(a)

[6]: 8

[11]: for i in np.arange(0, 8): # np.size(a) = 8
          print("a[", i, "] = ", a[i]);

a[ 0 ] =  7
a[ 1 ] =  2
a[ 2 ] =  7
a[ 3 ] =  6
a[ 4 ] =  9
a[ 5 ] =  0
a[ 6 ] =  3
a[ 7 ] =  0

[9]: np.arange(0, np.size(a))

[9]: array([0, 1, 2, 3, 4, 5, 6, 7])

[ ]:
```

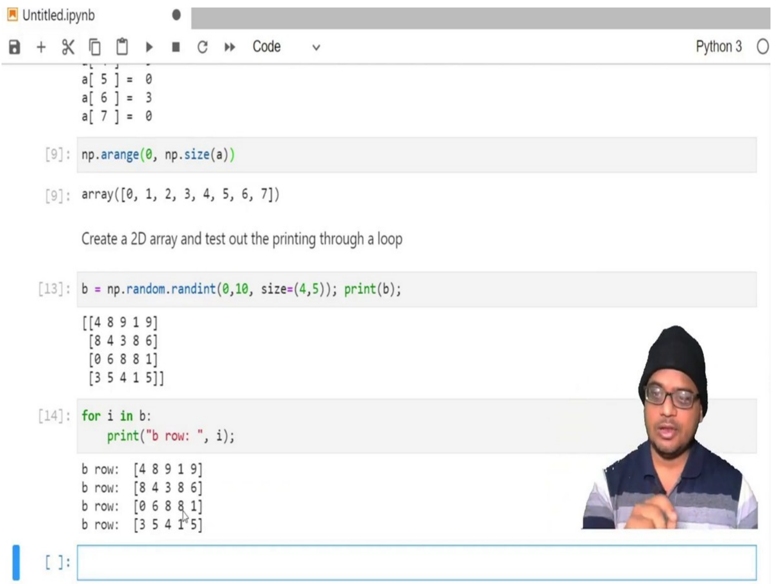
So, np.size() is a way of not hard coding the length of the grid. I can effectively do this as well this would output the same number. But usually it is not or rather quite often it is not a good idea to hard code the number ok. Let me create a 2 dimensional array.

(Refer Slide Time: 07:08)



```
[6]: np.size(a)

[6]: 8

[11]: for i in np.arange(0, np.size(a)): # np.size(a) = 8
          print("a[", i, "] = ", a[i]);

a[ 0 ] = 7
a[ 1 ] = 2
a[ 2 ] = 7
a[ 3 ] = 6
a[ 4 ] = 9
a[ 5 ] = 0
a[ 6 ] = 3
a[ 7 ] = 0

[9]: np.arange(0, np.size(a))

[9]: array([0, 1, 2, 3, 4, 5, 6, 7])

[ ]: b =
```

(Refer Slide Time: 07:14)



```
a[ 5 ] = 0
a[ 6 ] = 3
a[ 7 ] = 0

[9]: np.arange(0, np.size(a))

[9]: array([0, 1, 2, 3, 4, 5, 6, 7])
```

Create a 2D array and test out the printing through a loop

```
[13]: b = np.random.randint(0,10, size=(4,5)); print(b);

[[4 8 9 1 9]
 [8 4 3 8 6]
 [0 6 8 8 1]
 [3 5 4 1 5]]

[14]: for i in b:
          print("b row: ", i);

b row:  [4 8 9 1 9]
b row:  [8 4 3 8 6]
b row:  [0 6 8 8 1]
b row:  [3 5 4 1 5]

[ ]:
```

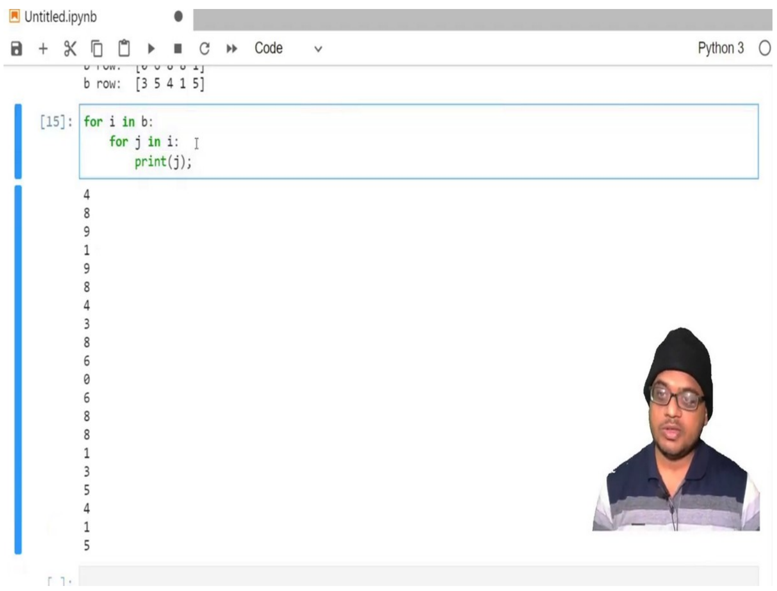
So, let me say b; so let me create a markdown ok. So, let me create a random array. So, b=np.random.randint(0,10, size =(4,5));

So, let me print out b. So, this is the matrix b, it contains 5 columns and 4 rows. So, how do we print out each element of an array? Now remember just because I am printing out each element of an array does not mean that iterators or loops should be used for that. You can use loops to achieve some computation inside the loop as well. In this particular case we are doing it just to showcase or highlight how a loop works.

So, we can do for i in b print(i). So, what do this will do is; it will print out all the elements in a given row in b. So, let me print out something pretty ok. So, it has printed me; 4 times and it is printed me the entire row.

Now, suppose I do not want to print the entire row I want to print out each element. So, remember; that i now contains all the elements of the row because you are printing i it means that you are essentially printing all the rows. So, now, you have to loop over all the elements of i.

(Refer Slide Time: 09:37)

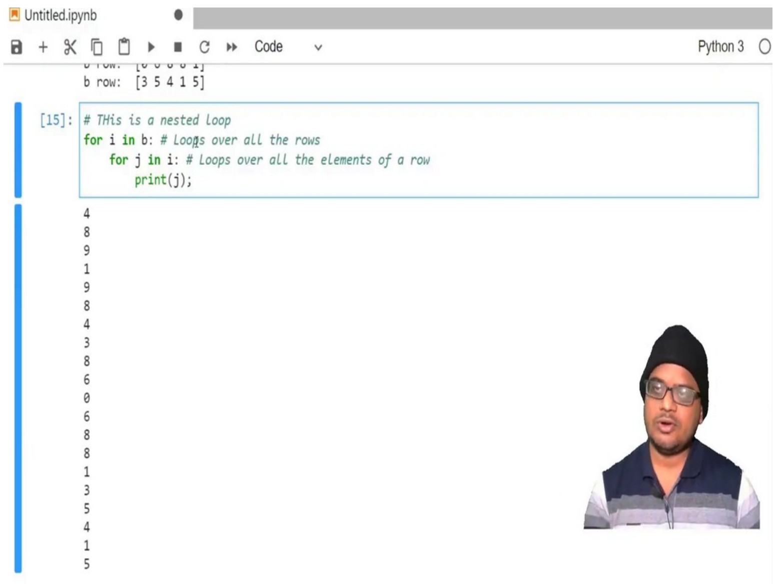


```
b row:  [3 5 4 1 5]

[15]: for i in b:
          for j in i:
              print(j);

4
8
9
1
9
8
4
3
8
6
0
6
8
8
1
3
5
4
1
5
```

So, how do we do that? We say for i in b for j in i print(j). What this will do is; j will loop over all the elements of i where i is the row of the matrix b and it will print everything one by one. Ok there you have a these are all the elements of b.

In fact, we can use another inbuilt function of the numpy module to print all the elements without having to resort to this kind of a nested loop, this kind of a double loop in which there is an outer loop and then there is a inner loop is called as a nested loop.

(Refer Slide Time: 10:28)


Untitled.ipynb

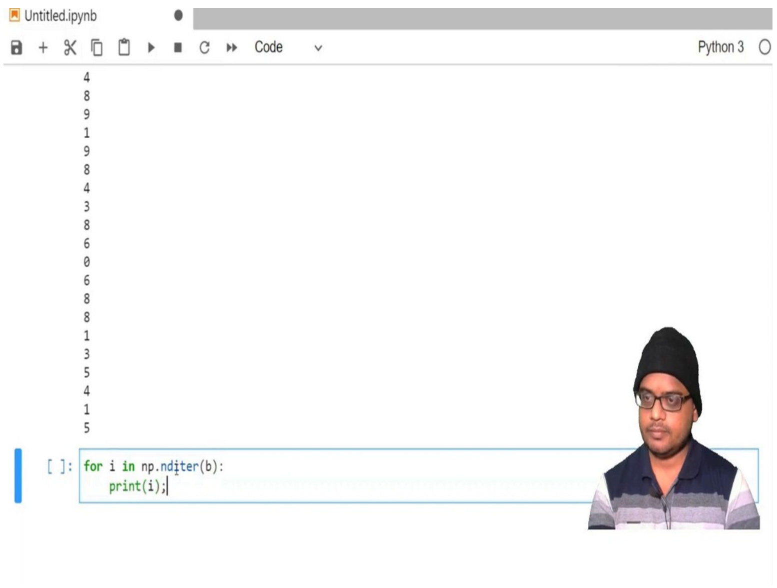
```
b row:  [3 5 4 1 5]
```

```
[15]: # THis is a nested loop
      for i in b: # Loops over all the rows
          for j in i: # Loops over all the elements of a row
              print(j);
```

```
4
8
9
1
9
8
4
3
8
6
0
6
8
8
1
3
5
4
1
5
```

In fact, this loops over all the rows and this loops over all the elements of a row; so this is what is happening. So, it picks up one row; it prints all the element of that row and it goes to the next row, it prints all the elements of that row, goes to the next row prints all the elements of the row and so on.

(Refer Slide Time: 11:10)

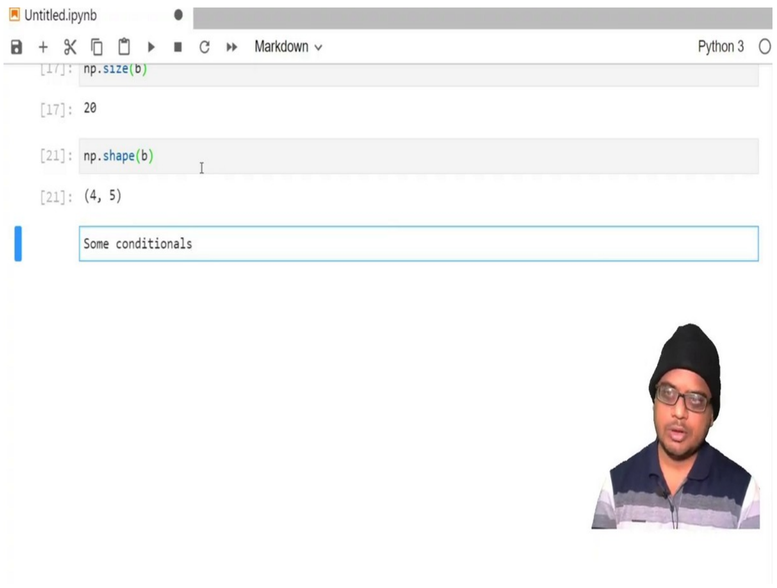

Untitled.ipynb

```
4
8
9
1
9
8
4
3
8
6
0
6
8
8
1
3
5
4
1
5
```

```
[ ]: for i in np.nditer(b):
         print(i);
```

So, instead of having this kind of a nested loop structure we can simply call the function. So, for i in np.nditer(b) print(i); so there you have it. So, np.nditer is an n dimensional iteration over all the elements of b. Because b contains how many rows and how many columns it is python can actually look into that create the nested loop for you and run the loop over all the elements you do not have to do a nested loop for that.

(Refer Slide Time: 11:48)



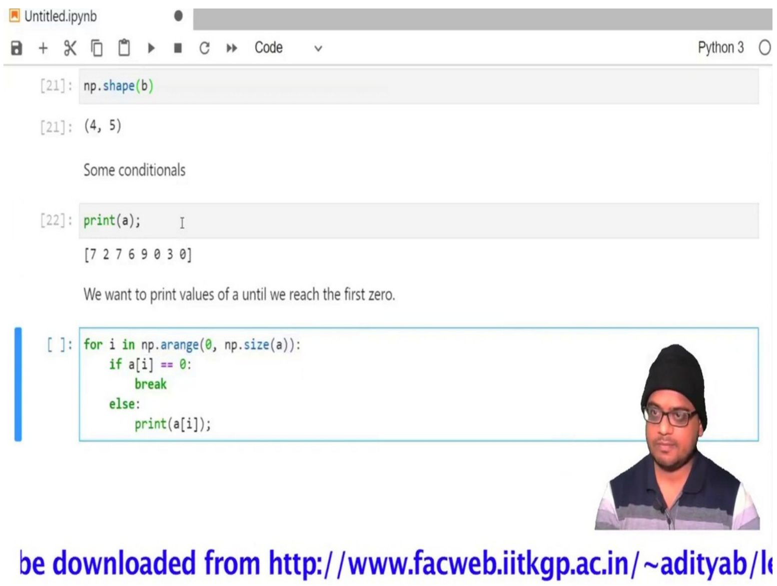
```
[17]: np.size(b)
[17]: 20
[21]: np.shape(b)
[21]: (4, 5)
```

Some conditionals

So, let us quickly verify; so let us print np.size(b). So, the total size of b is 20 because 4 cross 5 is 20 np.shape(b) its 4 rows and 5 columns. So, this is how you can query the matrix size and the number of elements.

Let us now proceed to write down some conditionals. In particular we are going to look at how we can check for certain values and how we can break loops. Because this is quite important when you are doing scientific computation.

(Refer Slide Time: 12:35)



```
[21]: np.shape(b)

[21]: (4, 5)
```

Some conditionals

```
[22]: print(a);

      [7 2 7 6 9 0 3 0]
```

We want to print values of a until we reach the first zero.

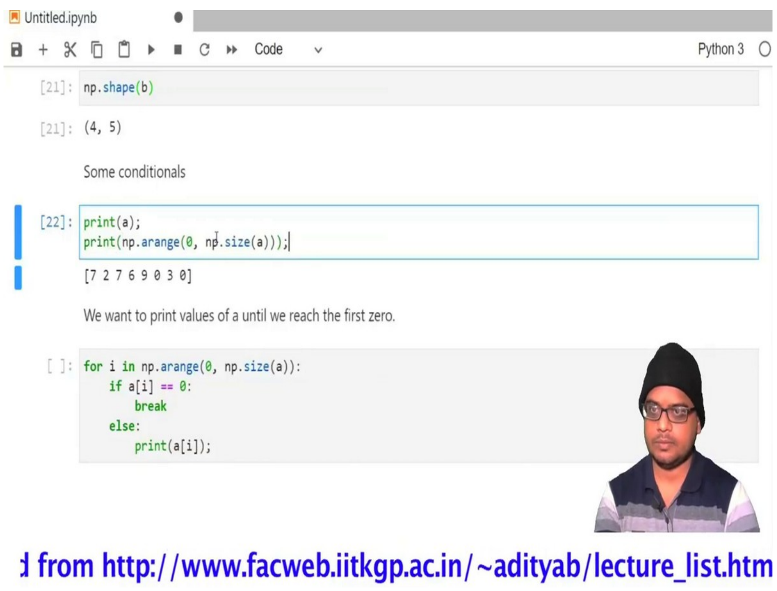
```
[ ]: for i in np.arange(0, np.size(a)):
         if a[i] == 0:
             break
         else:
             print(a[i]);
```

be downloaded from http://www.facweb.iitkgp.ac.in/~adityab/l

So, let us print out the value of a for reference ok, so a is this array. Now, suppose I want to print out all the values of a upon the point it reaches its first 0. So, how do we do that? So, let us first construct the loop for i in a. Or infact let me do it the c way or the MATLAB way; so for i in np.arange(0,np.size(a)).

So, now if a[i ]= 0; so this is a conditional which checks whether a[i] = 0 then we have to break this loop. So, once it encounters a break condition it will break out of that loop completely, it will not execute the loop any further; else print a[i]. So, what is going on over here? So, whatever we are doing is we are looping over all the indices inside this. So, let me just print this for reference.

(Refer Slide Time: 14:16)



```
[21]: np.shape(b)

[21]: (4, 5)
```

Some conditionals

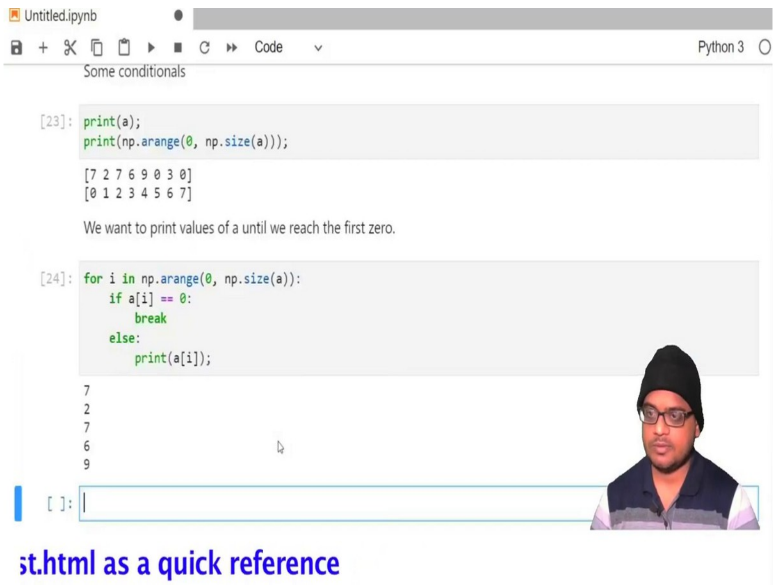
```
[22]: print(a);
      print(np.arange(0, np.size(a)));

      [7 2 7 6 9 0 3 0]
```

We want to print values of a until we reach the first zero.

```
[ ]: for i in np.arange(0, np.size(a)):
         if a[i] == 0:
             break
         else:
             print(a[i]);
```

d from http://www.facweb.iitkgp.ac.in/~adityab/lecture_list.htm

(Refer Slide Time: 14:22)



Some conditionals

```
[23]: print(a);
      print(np.arange(0, np.size(a)));

      [7 2 7 6 9 0 3 0]
      [0 1 2 3 4 5 6 7]
```

We want to print values of a until we reach the first zero.

```
[24]: for i in np.arange(0, np.size(a)):
          if a[i] == 0:
              break
          else:
              print(a[i]);

      7
      2
      7
      6
      9

[ ]:
```

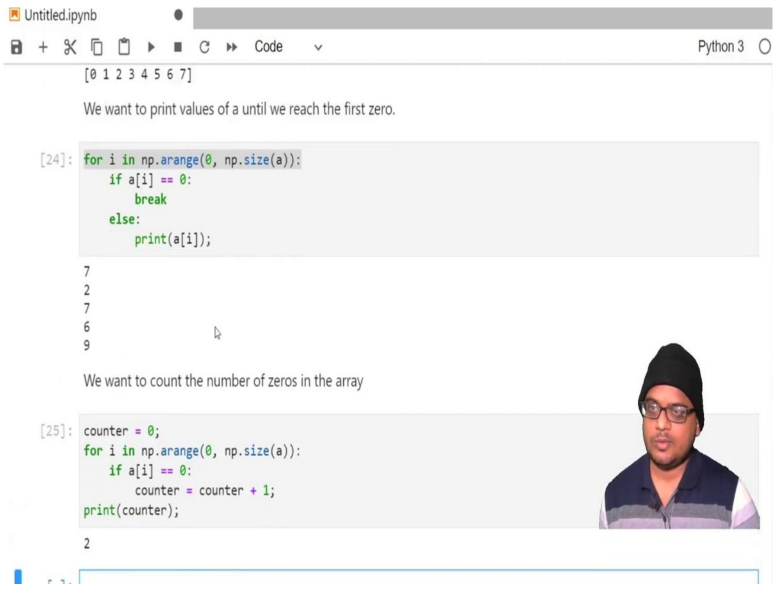
st.html as a quick reference

So, we are looping over all these elements. So, which checks it checks whether a [i] is equal to 0 or not. So, whether a[0]=0 no, a [1]=0 no, a[2] and so on. So, we see that a[5] = 0 ok, a[5] = 0. So, it should break the loop.

So, let us see what happens. So, what do we have 7, 2, 7, 6, 9, 7, 2, 7, 6, 9 and the moment we reach a 5 it will check, if a[5]==0? Yes a[5]==0; so it will break the loop. So, this is how you can have a conditional inside your loop to break the loop.

Now suppose you are interested to know how many. So, this is a very small loop so you can tell pretty much straight away that there has been 5 number of elements prior to reaching its first 0. But suppose you want to count the total number of times or you want to; suppose I want to count how many zeros are there in the array a; yeah, let us do that.

(Refer Slide Time: 15:47)



```
[0 1 2 3 4 5 6 7]
```

We want to print values of a until we reach the first zero.

```
[24]: for i in np.arange(0, np.size(a)):
          if a[i] == 0:
              break
          else:
              print(a[i]);
```

```
7
2
7
6
9
```

We want to count the number of zeros in the array

```
[25]: counter = 0;
      for i in np.arange(0, np.size(a)):
          if a[i] == 0:
              counter = counter + 1;
      print(counter);
```

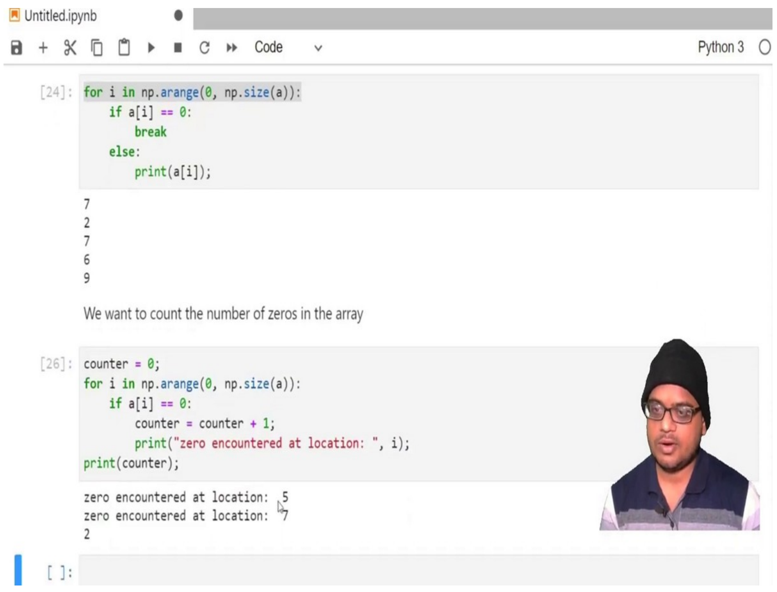
```
2
```

So, let me copy this template. So, we want to number of zeors in the array; so yeah. So, we will write; so we have to first define a counter we will define counter=0; it means that I have a counter which will increment each time I encounter 0.

So, if a[i]=0 right then counter will be counter+1, else it has to do nothing; so let us run this. So in fact at the end of the loop we should print counter. So, it gives an answer 2; so it has counted it twice.

In fact, suppose we want to modify this loop to tell us at which indices 0 has occurred. We want to tell so we know that it has occurred at index number 5 and index number 7, so how do we do that?

(Refer Slide Time: 17:08)



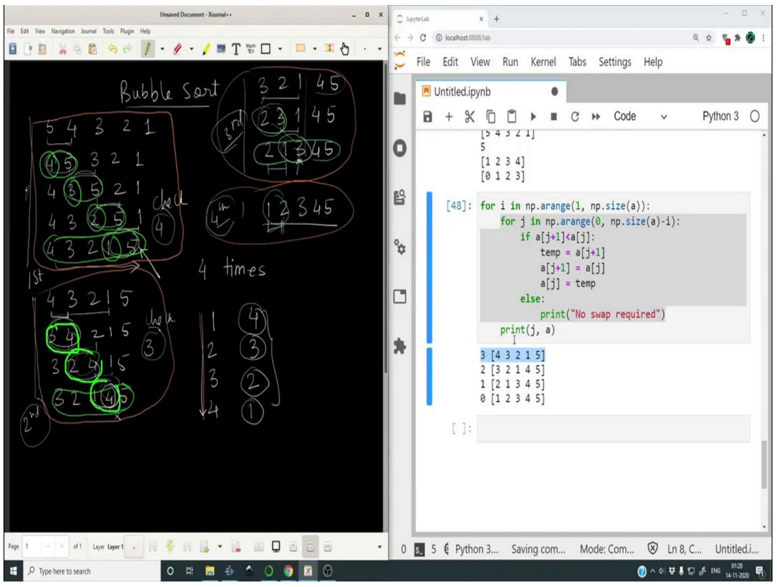
```
[24]: for i in np.arange(0, np.size(a)):
          if a[i] == 0:
              break
          else:
              print(a[i]);

7
2
7
6
9
```

We want to count the number of zeros in the array

```
[26]: counter = 0;
      for i in np.arange(0, np.size(a)):
          if a[i] == 0:
              counter = counter + 1;
              print("zero encountered at location: ", i);
      print(counter);

zero encountered at location:  5
zero encountered at location:  7
2
```

So, let me create c index so let me initialize it as 0. So, the moment encounters a[i]=0 we will print zero encountered at location and then we will print out the value of i. In fact, we did not require c index if we wanted to store something else we would; so we just print out the index. So, let me run this and see what happens. So, zero encountered at 5 and at 7. Let us now use our knowledge acquired so far to create a simple program for bubble sort.

(Refer Slide Time: 18:00)



```
[5 4 3 2 1]
5
[1 2 3 4]
[0 1 2 3]

[48]: for i in np.arange(1, np.size(a)):
          for j in np.arange(0, np.size(a)-i):
              if a[j+1]<a[j]:
                  temp = a[j+1]
                  a[j+1] = a[j]
                  a[j] = temp
              else:
                  print("No swap required")
          print(j, a)

3 [4 3 2 1 5]
2 [3 2 1 4 5]
1 [2 1 3 4 5]
0 [1 2 3 4 5]
```

So, what is bubble sort? So, imagine we have an array something like this; 5, 4, 3, 2, 1 and we want to sort it in ascending order. So, we first look at these 2 elements. So, because this element is smaller than this element we will swap.

Then we will look at these 2 elements because this is smaller than this we will again swap. Now between these 2 elements this is smaller than this so again we will swap. Now between these 2 elements this is smaller than this so we will swap. So, notice how the largest element is rising through the array like a bubble; now, so this is the first loop.

In the second sort of iteration that we will have we will start with this. So, now, because we know that the largest element has gone all the way to the end, we need to only loop between these many elements and we have to perform the same checks. So, let us check between these two 3 smaller than 4; so this will swap so 3, 4, 2, 1, 5.

Now between these two so this will be 3, 2, 4, 1, 5. Now between these two; so this will be 3 4 2 1 sorry 3 2 1 4 5. Now between these two 5 is larger than 4; so there will be nothing no swap over here ok. So, in the second set of iterations we have effectively transferred the next largest element to the next to last location. So, it has risen like a bubble, but it is not going all the way to the end because it is not the largest element.

So, now, what? So, now, we have 3, 2, 1, 4, 5. Now we have to loop over only these elements and make the same pairs of comparison. So, 2 is smaller than 3 so we will swap; then we will check this so 1 is smaller than 3; so we will swap and we end because we only have to loop over these many elements. Because these are already at the end through this these two loops we have already obtained them sorted.

So, now this third largest element has also gone to the end. Now we have to perform a check between these two elements. So, what is this? So, 1 is smaller than 2; so we will perform a swap. So, finally, we will get 1, 2, 3, 4, 5. So, this was the third iterate and this was the fourth iterate. So, this particular algorithm of sorting of an array is called as bubble sort because each subsequent larger largest number rises through the array like a bubble like a bubble rises in water.

So, as you can imagine since we have 5 elements we have to loop we have to perform this check 4 times. And for each time so 1, 2, 3, 4 and inside each loop how many pairs do we have? So, over here we had 1 pair, 2 pairs, 3 and 4 so we had 4 pairs over here. Over here we had sorry we did not have 4 we had only 3 because this was the starting array. So, we did a swap once twice thrice and then 4 yeah.

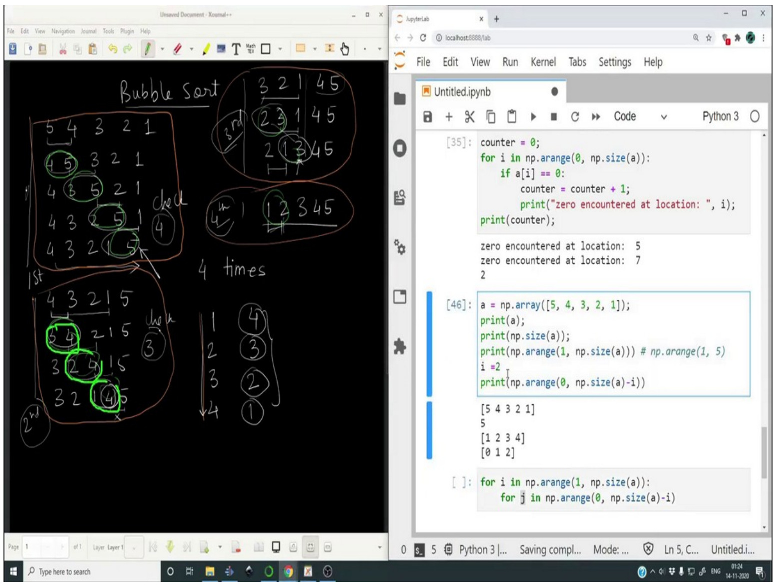
So, then what did we do? We took this array we did a swap 1, 2, 3 so we did a swap 3 times we did a check 3 times. Over here we did a check 1, 2 twice and lastly we did a check once.

So, depending on what you are doing so if you look; so in the first outermost loop you did 4 checks, in the second outermost loop you did 3 checks, in the third iteration you did 2 checks and in the fourth iteration you did 1 check. You see that this sum remains constant it is equal to the number of elements.

So, this gives an idea that you have to perform the operation 4 you have to loop over the entire array 4 times, but each time you have to check lesser and lesser number of pairs ok. So, once you have achieved the highest element all the way to the end you do not need to check it till the last element, you can check it till the second last element.

Once you have sorted the second last element you have to sort it till the third last element and so on. So, that is how you save on some computations. So, let us now code whatever we have seen in that algorithm of bubble sort in python. So, the first thing is to create that particular array.

(Refer Slide Time: 24:29)



```
[35]: counter = 0;
      for i in np.arange(0, np.size(a)):
          if a[i] == 0:
              counter = counter + 1;
              print("zero encountered at location: ", i);
      print(counter);

      zero encountered at location:  5
      zero encountered at location:  7
      2

[46]: a = np.array([5, 4, 3, 2, 1]);
      print(a);
      print(np.size(a));
      print(np.arange(1, np.size(a))) # np.arange(1, 5)
      i =2
      print(np.arange(0, np.size(a)-i))

      [5 4 3 2 1]
      5
      [1 2 3 4]
      [0 1 2]

[ ]:  for i in np.arange(1, np.size(a)):
          for j in np.arange(0, np.size(a)-i)
```

Usually in your program you will have a generated array by some experiment or some numerical means, but for this particular example let us create this particular array. So, a = np.array([5, 4, 3, 2, 1]).

So, let me print a let me print the size of a as well, we will require this later on. So, a is 5, 4, 3, 2, 1 and the size of array a is 5. So, let me now create the outermost loop. So, what is the outermost loop? So, this particular orange box is the first loop or the first element of the outermost loop, this is the second element of the outermost loop, this is the third element of the outermost loop and the fourth ok.

So, for 5 elements it loops from 1, 2, 3, 4 and I could have simply done a loop from 1 to 4, but I want to generalize it I don't want to hard code that value of fourth. So, what I will do is; I will query the size of a and accordingly set the loops. So, I will do for i in np.arange(1, np.size( a)).

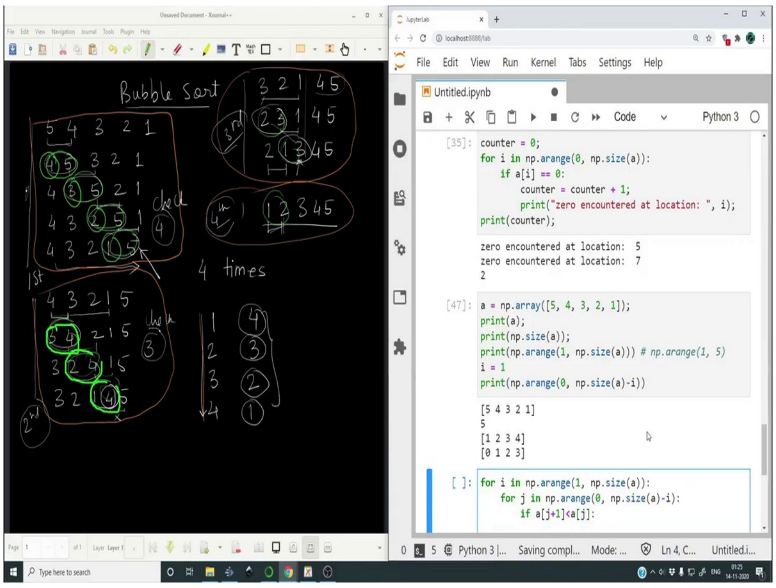
So, if we want to see what this loop sequence will actually look like let us print it out. So, it will be 1, 2, 3, 4 ok. So, np.arange(1,5). So, essentially this means np.arange() which goes from 1 and ends to ends till 4 because np.arange() excludes the final value, it increments by 1 and it excludes the final value. So, it will be 1, 2, 3, 4 it will not include 5. If you wanted to include 5 we have to input 6 over here ok.

So, the next loop will be the inner loop. So, inside each of these inside each of these orange boxes we have 1, 2, 3, 4, 1, 2, 3, 1, 2 and 1. So, as the outermost array increases the number of checks we have to do inside each array that those number of checks they reduce.

So, and how do they reduce? It reduces such that 1 plus 4 remains constant 2 plus 3 remains constant 3 plus 2 remains constant and so on. So, we will make over here for j in np.arange(0,np.size(a)-i) ok. So, in fact, this should be minus i it should not be minus 1 minus i. So, when i is equal to 2 ok. So, let us just print it out as well let me set i = 2 and let me print this out.

So, when i = 2; i would loop j over 0, 1, 2. So, this is i= 2. So, this is the zeroth loop this is 1 this is 2. So, I would do the check 3 times let me change it to i = 1.

(Refer Slide Time: 28:37)

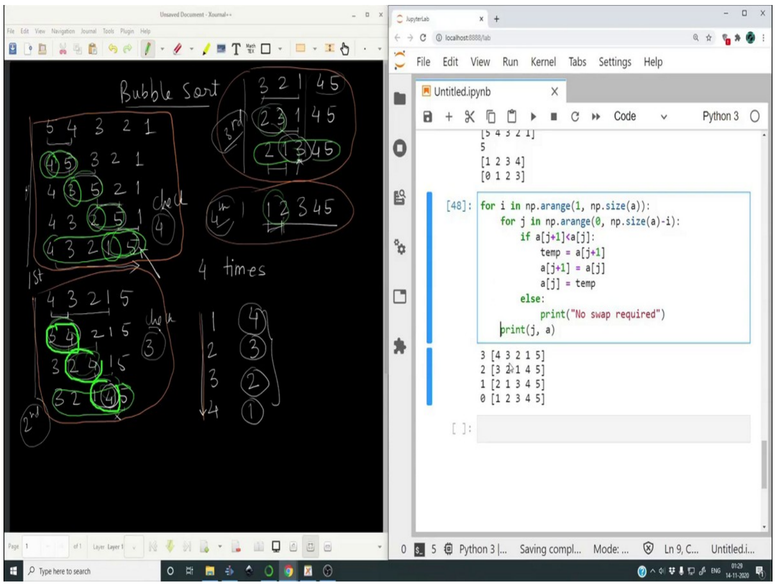


```
[35]: counter = 0;
      for i in np.arange(0, np.size(a)):
          if a[i] == 0:
              counter = counter + 1;
              print("zero encountered at location: ", i);
      print(counter);

      zero encountered at location:  5
      zero encountered at location:  7
      2

[47]: a = np.array([5, 4, 3, 2, 1]);
      print(a);
      print(np.size(a));
      print(np.arange(1, np.size(a))) # np.arange(1, 5)
      i = 1
      print(np.arange(0, np.size(a)-i))

      [5 4 3 2 1]
      5
      [1 2 3 4]
      [0 1 2 3]

[ ]:  for i in np.arange(1, np.size(a)):
          for j in np.arange(0, np.size(a)-i):
              if a[j+1]<a[j]:
```

So, let me run this 1, 2, 0, 1, 2, 3. So, this is 0, this is 1, this is 2, this is 3 and actually it corroborates with the index this is index number 0 index number 1, index number 2, index number 3. So, this is how we can achieve the nested sequence that we have one is the orange box and one is the green box.

So, now that we have our nested loop structure we will check if a[j+1] if it is smaller than a[j]? Then what do we have to do? We have to do the following; we have to swap a[j+1] and a[j].

(Refer Slide Time: 29:32)



```
      [5 4 3 2 1]
      5
      [1 2 3 4]
      [0 1 2 3]

[48]: for i in np.arange(1, np.size(a)):
          for j in np.arange(0, np.size(a)-i):
              if a[j+1]<a[j]:
                  temp = a[j+1]
                  a[j+1] = a[j]
                  a[j] = temp
              else:
                  print("No swap required")
          print(j, a)

      3 [4 3 2 1 5]
      2 [3 2 1 4 5]
      1 [2 1 3 4 5]
      0 [1 2 3 4 5]

[ ]:
```

So, the way to do it is we will first assign a[j+1] to a temporary buffer. We will update the value of a[j] with a[j+1] and then we will update the value of a[j] with temp. So, this ensures that none of the variables are overwritten. So, temporary is just like a temporary buffer. So, if you just think about it will be clear. So, that is it that is pretty much the loop.

So yeah so the other thing is if this happens you execute this, if this condition is true it will execute all this. And if we wanted to do something else so we will write else you can say print("no swapping required") ok. So, if that condition is not satisfied we do not need to swap anything its already sorted yeah.

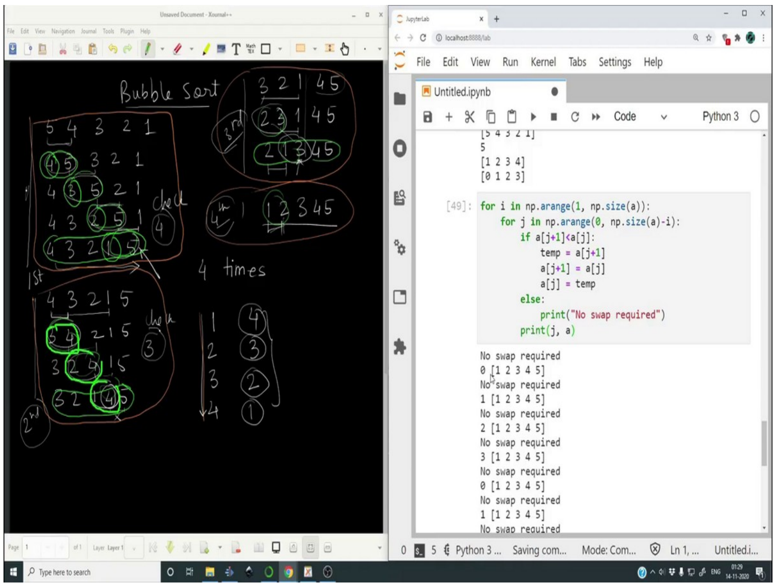
So, let us exit the scope of the if else statement and let me write over here let me print it out. So, let me print the value of j and the array a. So, that bubble sort behavior would be apparent. Lastly, no yeah thats it; so let us do this. So, let me run this cell and let us see what happens.

Actually we are out of the scope of this function. So, I am glad this error came up. So, the function of the if else statement is for all the statements which are at an indentation from this particular if. So, all these statements are inside this scope this statement is inside this scope, this particular statement has to be inside the scope of this for loop. So, we will press tab and put it over here. So, now for each time this inner check happens we would have printed the value of j and the evolution of the array a ok.

So, previously it was something like this. So, after all the inner checks are done it has printed the final sorted list after each inner check. So, look at this 4, 3, 2, 1, 5. So, this checks out with this final result of this first loop.

Then we have 3, 2, 1, 4, 5 this checks out with the final result of this particular inner loop this is the final result of that particular inner loop and this also checks out. So, we do not want to print out only the final output we want to print it out dynamically we want to look at how a has evolved over all the iterations. So, we have to move the print statement into this particular loop. So, it has to have an indentation. So, indentation is obtained by pressing tab ok.

(Refer Slide Time: 33:05)



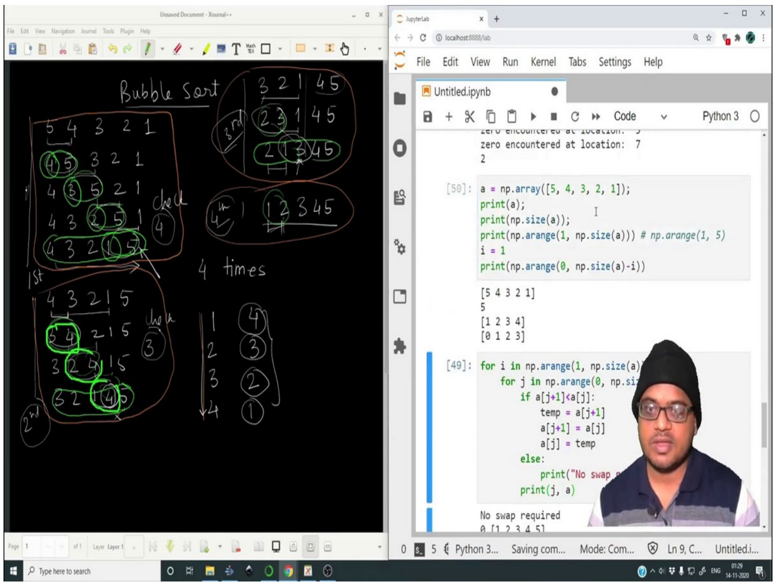
```
[5 4 3 2 1]
5
[1 2 3 4]
[0 1 2 3]
```

```
[49]: for i in np.arange(1, np.size(a)):
          for j in np.arange(0, np.size(a)-i):
              if a[j+1]<a[j]:
                  temp = a[j+1]
                  a[j+1] = a[j]
                  a[j] = temp
              else:
                  print("No swap required")
              print(j, a)
```

```
No swap required
0 [1 2 3 4 5]
No swap required
1 [1 2 3 4 5]
No swap required
2 [1 2 3 4 5]
No swap required
3 [1 2 3 4 5]
No swap required
0 [1 2 3 4 5]
No swap required
1 [1 2 3 4 5]
No swap required
```

So, let me press let me run this loop. So, a was already sorted so we have to run all the cells again.

(Refer Slide Time: 33:19)



```
zero encountered at location:  7
2
```

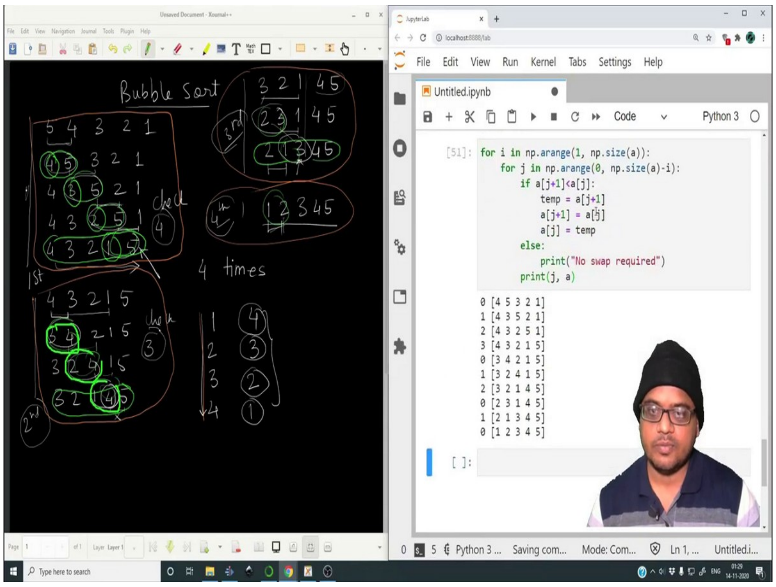
```
[50]: a = np.array([5, 4, 3, 2, 1]);
      print(a);
      print(np.size(a));
      print(np.arange(1, np.size(a))) # np.arange(1, 5)
      i = 1
      print(np.arange(0, np.size(a)-i))
```

```
[5 4 3 2 1]
5
[1 2 3 4]
[0 1 2 3]
```

```
[49]: for i in np.arange(1, np.size(a)
          for j in np.arange(0, np.si
              if a[j+1]<a[j]:
                  temp = a[j+1]
                  a[j+1] = a[j]
                  a[j] = temp
              else:
                  print("No swap
              print(j, a)
```

```
No swap required
```

(Refer Slide Time: 33:20)



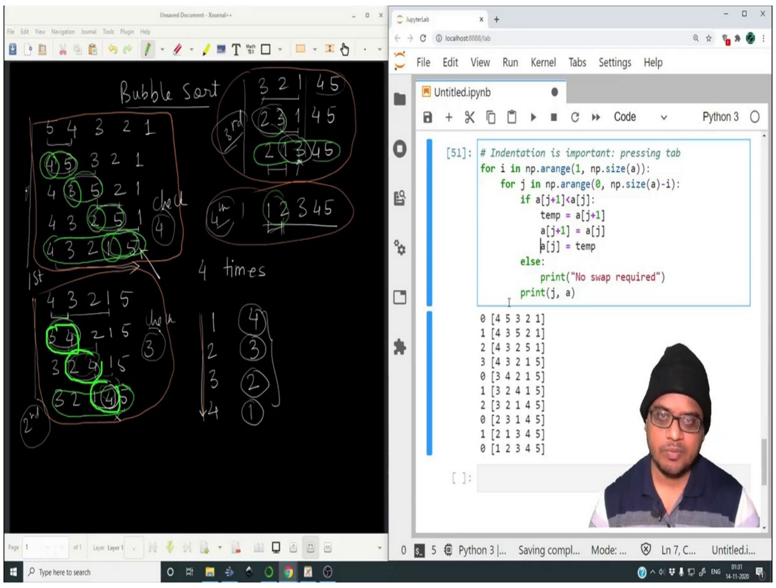
```
[51]: for i in np.arange(1, np.size(a)):
          for j in np.arange(0, np.size(a)-i):
              if a[j+1]<a[j]:
                  temp = a[j+1]
                  a[j+1] = a[j]
                  a[j] = temp
              else:
                  print("No swap required")
              print(j, a)
```

```
0 [4 5 3 2 1]
1 [4 3 5 2 1]
2 [4 3 2 5 1]
3 [4 3 2 1 5]
0 [3 4 2 1 5]
1 [3 2 4 1 5]
2 [3 2 1 4 5]
0 [2 3 1 4 5]
1 [2 1 3 4 5]
0 [1 2 3 4 5]
```

So, let me run this cell let me run this cell again ok. So, 0, 1, 2, 3 this is the first orange box inside the first orange box look 4 and 5 have been swapped. So, we are at this step 3 and 5 have been swapped 2 and 5 have been swapped 1 and 5 have been swapped.

This is the second orange box this one 3 and 4 swapped 2 and 4 swap 1 and 4 swapped. Then this is the and this particular thing is this particular orange box you have 2 swaps and this is the last thing there is nothing to swap ok. So, this is how we can encode a bubble sort and through this we have looked at several things the let me just summarize it over here; indentation is important.

(Refer Slide Time: 34:12)

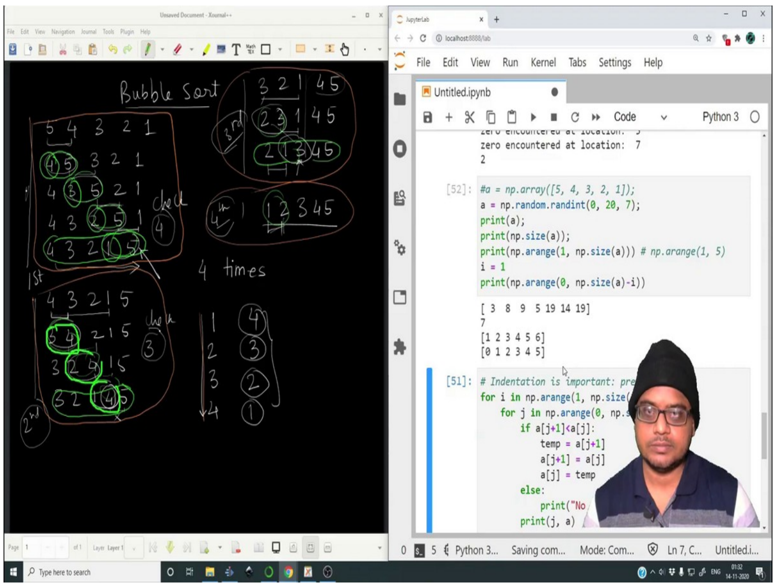


```
[51]: # Indentation is important: pressing tab
      for i in np.arange(1, np.size(a)):
          for j in np.arange(0, np.size(a)-i):
              if a[j+1]<a[j]:
                  temp = a[j+1]
                  a[j+1] = a[j]
                  a[j] = temp
              else:
                  print("No swap required")
              print(j, a)

      0 [4 5 3 2 1]
      1 [4 3 5 2 1]
      2 [4 3 2 5 1]
      3 [4 3 2 1 5]
      0 [3 4 2 1 5]
      1 [3 2 4 1 5]
      2 [3 2 1 4 5]
      0 [2 3 1 4 5]
      1 [2 1 3 4 5]
      0 [1 2 3 4 5]

[ ]:
```

So, indentation is achieved by pressing tab not by pressing spaces this is a single tab. So, whatever is indented from this particular for loop will be executed inside that for loop. So, for example, this for loop is inside this so it is indented, whatever is indented inside this would be executed inside this innermost for loop. So, all these things are indented inside this. So, inside each loop this will be executed. Whatever is indented for this if statement would be executed inside this if block.

So, suppose I were to remove this. So, the if statement would only execute these 2 commands this is outside the if statement. So, doing this would indent it back into the if statement in programming languages like c. This is not a big deal because you have curly braces blocking out sections of the code. In this course I am already assuming that you have some basic knowledge of programming, otherwise it would be quite difficult to jump straight into python.

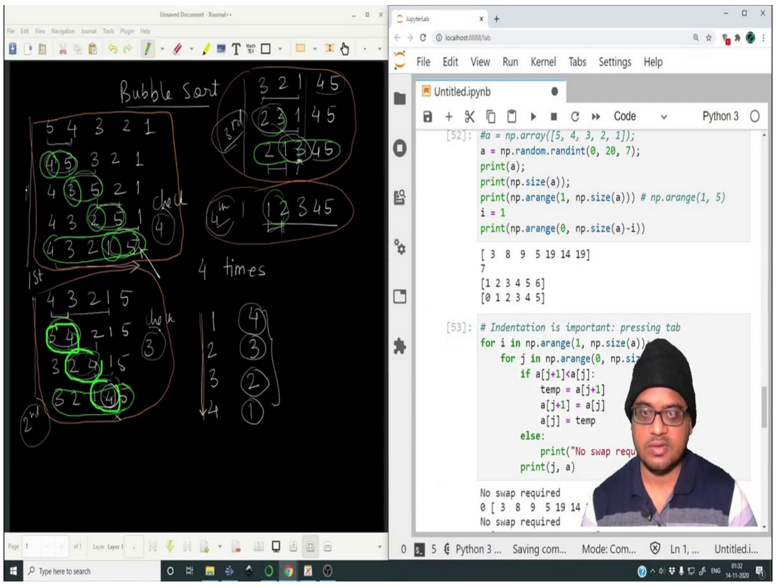
(Refer Slide Time: 35:39)



```
zero encountered at location:  7
2

[52]: #a = np.array([5, 4, 3, 2, 1]);
      a = np.random.randint(0, 20, 7);
      print(a);
      print(np.size(a));
      print(np.arange(1, np.size(a))) # np.arange(1, 5)
      i = 1
      print(np.arange(0, np.size(a)-i))

      [ 3  8  9  5 19 14 19]
      7
      [1 2 3 4 5 6]
      [0 1 2 3 4 5]

[51]: # Indentation is important: pre
      for i in np.arange(1, np.size(
          for j in np.arange(0, np.s
              if a[j+1]<a[j]:
                  temp = a[j+1]
                  a[j+1] = a[j]
                  a[j] = temp
              else:
                  print("No
              print(j, a)
```

Let me in fact, now generate a random array of integers instead of hard coding this particular example array we can choose a as np.random.randint(0,20,7). So, in opening value is 0 closing value is 20 and let us sample 7 values.

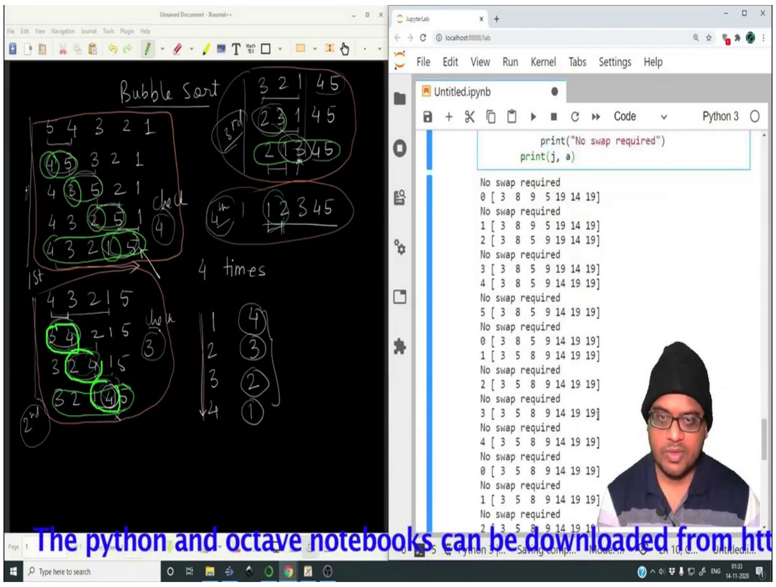
(Refer Slide Time: 36:05)



```
[52]: #a = np.array([5, 4, 3, 2, 1]);
      a = np.random.randint(0, 20, 7);
      print(a);
      print(np.size(a));
      print(np.arange(1, np.size(a))) # np.arange(1, 5)
      i = 1
      print(np.arange(0, np.size(a)-i))

      [ 3  8  9  5 19 14 19]
      7
      [1 2 3 4 5 6]
      [0 1 2 3 4 5]

[53]: # Indentation is important: pressing tab
      for i in np.arange(1, np.size(a))
          for j in np.arange(0, np.si
              if a[j+1]<a[j]:
                  temp = a[j+1]
                  a[j+1] = a[j]
                  a[j] = temp
              else:
                  print("No swap requ
              print(j, a)

      No swap required
      0 [ 3  8  9  5 19 14
      No swap required
```

So, let me run this cell and let me run this cell. So, let us look at this; so 3 and 8 its already sorted because 8 is larger than 3.

(Refer Slide Time: 36:15)



```
            print("No swap required")
        print(j, a)
```

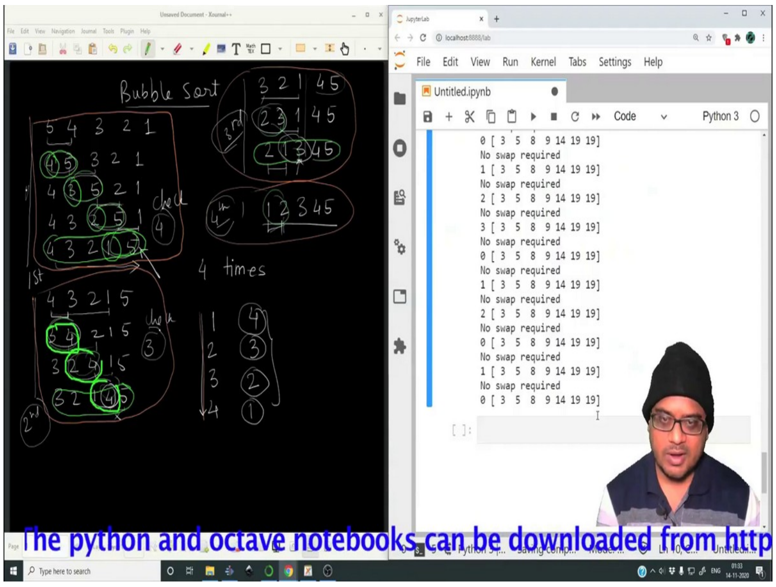
```
No swap required
0 [ 3  8  9  5 19 14 19]
No swap required
1 [ 3  8  9  5 19 14 19]
2 [ 3  8  5  9 19 14 19]
No swap required
3 [ 3  8  5  9 19 14 19]
4 [ 3  8  5  9 14 19 19]
No swap required
5 [ 3  8  5  9 14 19 19]
No swap required
0 [ 3  8  5  9 14 19 19]
1 [ 3  5  8  9 14 19 19]
No swap required
2 [ 3  5  8  9 14 19 19]
No swap required
3 [ 3  5  8  9 14 19 19]
No swap required
4 [ 3  5  8  9 14 19 19]
No swap required
0 [ 3  5  8  9 14 19 19]
No swap required
1 [ 3  5  8  9 14 19 19]
No swap required
2 [ 3  5  8  9 14 19 1
```

The python and octave notebooks can be downloaded from htt

So, no swap required great then we have this array between 8 and 9 there is no swap required great no swap required, between 9 and 5 there is a swap required. So, this is swapped.

Now, between 9 and 19 there is no swap required because once you have swapped 9 and 5 you end up with this configuration, but between 9 and 19 there is no swap required great. So, then this is how the entire loop works have a look at it try to implement other loop algorithms on your own. This is by far not the fastest sorting algorithm.

(Refer Slide Time: 36:56)

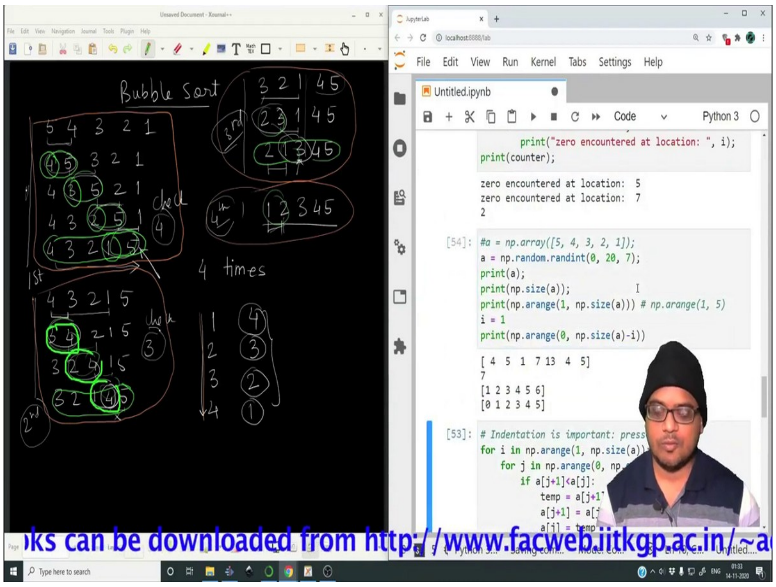


```
0 [ 3  5  8  9 14 19 19]
No swap required
1 [ 3  5  8  9 14 19 19]
No swap required
2 [ 3  5  8  9 14 19 19]
No swap required
3 [ 3  5  8  9 14 19 19]
No swap required
0 [ 3  5  8  9 14 19 19]
No swap required
1 [ 3  5  8  9 14 19 19]
No swap required
2 [ 3  5  8  9 14 19 19]
No swap required
0 [ 3  5  8  9 14 19 19]
No swap required
1 [ 3  5  8  9 14 19 19]
No swap required
0 [ 3  5  8  9 14 19 19]
```

The python and octave notebooks can be downloaded from http

In fact, you may wonder why bother doing all this? I mean there surely must be some inbuilt function in numpy which can help you do all this. And if you thought that you are correct there is a there is an inbuilt function to do that.

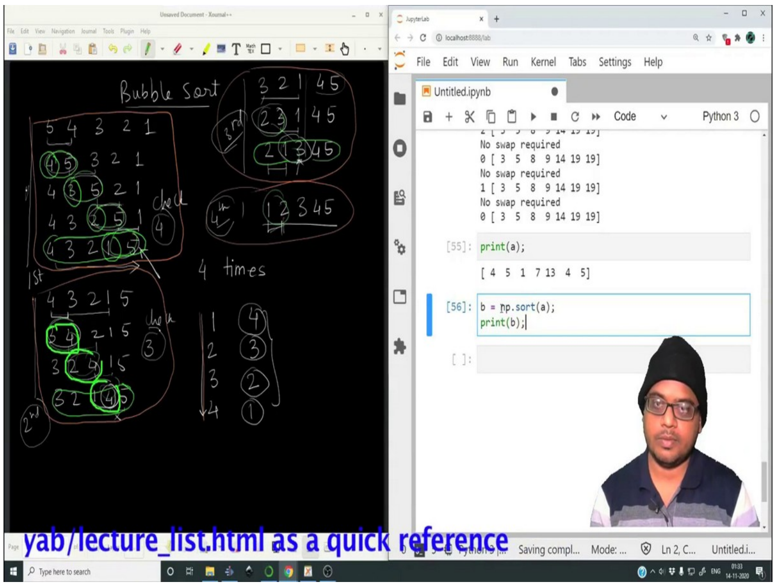
(Refer Slide Time: 37:11)



```
        print("zero encountered at location: ", i);
print(counter);

zero encountered at location:  5
zero encountered at location:  7
2

[54]: #a = np.array([5, 4, 3, 2, 1]);
a = np.random.randint(0, 20, 7);
print(a);
print(np.size(a));
print(np.arange(1, np.size(a))) # np.arange(1, 5)
i = 1
print(np.arange(0, np.size(a)-i))

[ 4  5  1  7 13  4  5]
7
[1 2 3 4 5 6]
[0 1 2 3 4 5]

[53]: # Indentation is important: press
for i in np.arange(1, np.size(a))
    for j in np.arange(0, np
        if a[j+1]<a[j]:
            temp = a[j+1
            a[j+1] = a[
            a[j] = temp
```

So, let me run this once to reinitialize the value of the array a; because running this cell this particular cell would swap everything inside a; so we do not want that.

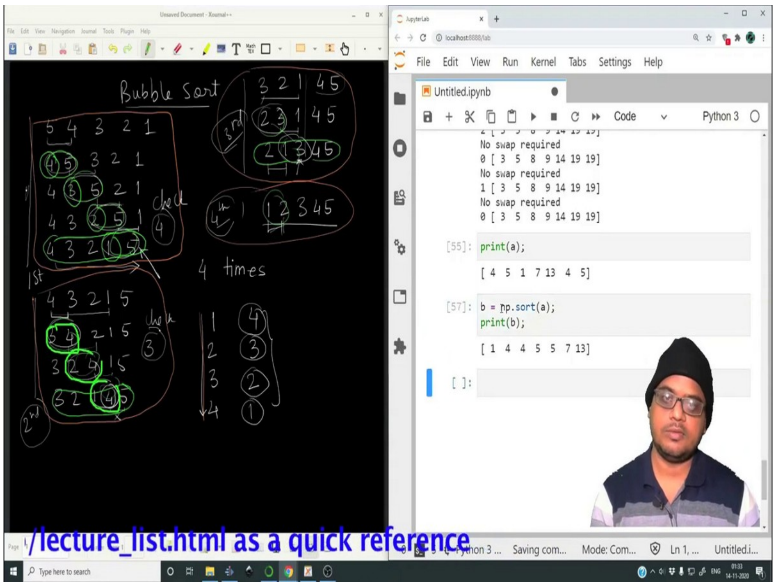
(Refer Slide Time: 37:22)



```
No swap required
0 [ 3  5  8  9 14 19 19]
No swap required
1 [ 3  5  8  9 14 19 19]
No swap required
0 [ 3  5  8  9 14 19 19]

[55]: print(a);

[ 4  5  1  7 13  4  5]

[56]: b = np.sort(a);
print(b);

[ ]:
```

So, now, let me print a. So, this is a, I have reinitialized with a random number; so each time I call that cell I will have a random array. So, then I can simply say b = np.sort(a) ok. So, let me print what b is; so this is the sorted array a.

(Refer Slide Time: 37:44)


Bubble Sort
File Edit View Run Kernel Tabs Settings Help
Untitled.ipynb
Code
Python 3
/lecture_list.html as a quick reference

```
No swap required
0 [ 3  5  8  9 14 19 19]
No swap required
1 [ 3  5  8  9 14 19 19]
No swap required
0 [ 3  5  8  9 14 19 19]
```

```
[55]: print(a);

[ 4  5  1  7 13  4  5]

[57]: b = np.sort(a);
      print(b);

[ 1  4  4  5  5  7 13]

[ ]:
```

So, to conclude in this particular lecture we have looked at conditionals, we have looked at loops and we have looked at how we can combine all those informations to create a very simple implementation of bubble sort. And lastly we have seen how to make use of the inbuilt function of numpy to sort arrays without having to write so much of code.

So, with this we conclude this particular lecture and I will see you next time; bye.