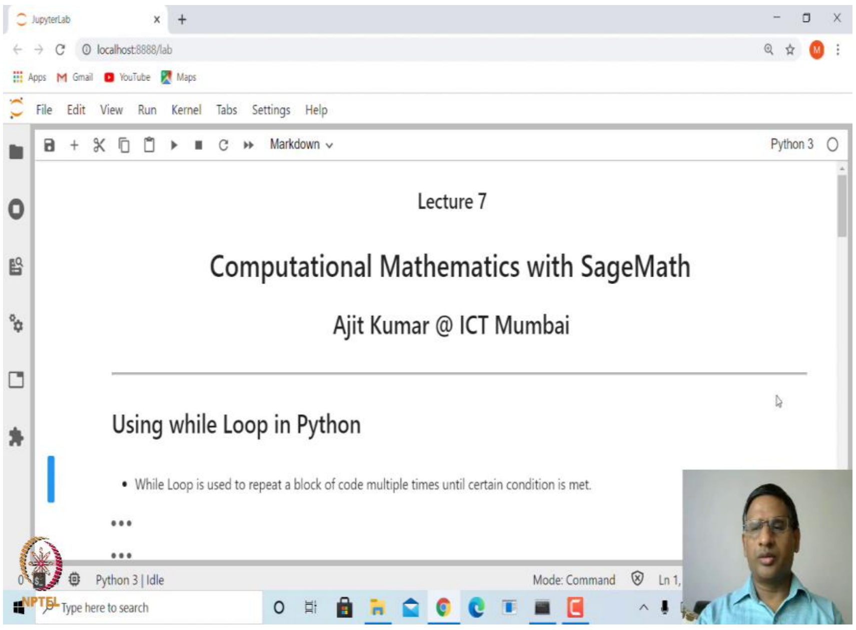
Computational Mathematics with Sagemath

Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Lecture – 08
While Loop in Python

(Refer Slide Time: 00:20)



Lecture 7

Computational Mathematics with SageMath

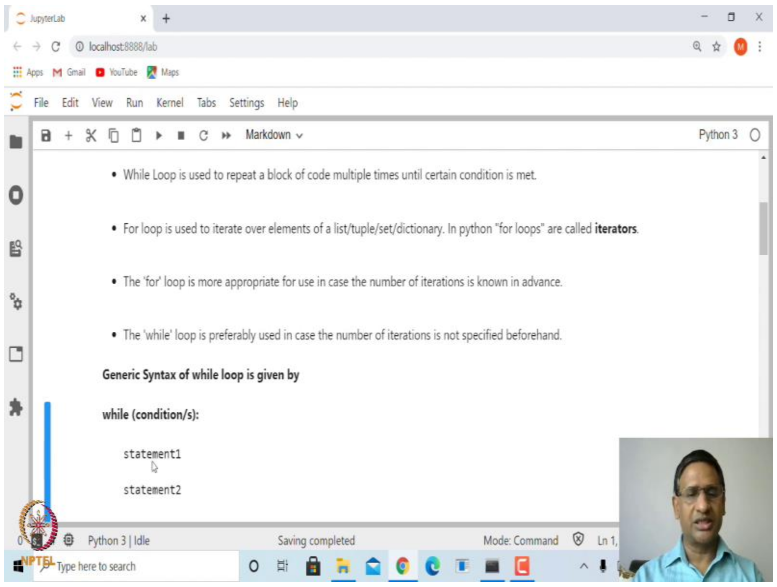
Ajit Kumar @ ICT Mumbai

Using while Loop in Python

- While Loop is used to repeat a block of code multiple times until certain condition is met.

Welcome to the 7th lecture on Computational Mathematics with SageMath. In this lecture, we are going to look at using While Loop in Python. While loop is used to repeat a block of codes multiple times until certain condition is met. So, we need to give the conditions and as long as that conditions, (it may be one condition or several conditions) are satisfied you need to execute a block of codes.

(Refer Slide Time: 00:51)



- While Loop is used to repeat a block of code multiple times until certain condition is met.
- For loop is used to iterate over elements of a list/tuple/set/dictionary. In python "for loops" are called **iterators**.
- The 'for' loop is more appropriate for use in case the number of iterations is known in advance.
- The 'while' loop is preferably used in case the number of iterations is not specified beforehand.

Generic Syntax of while loop is given by

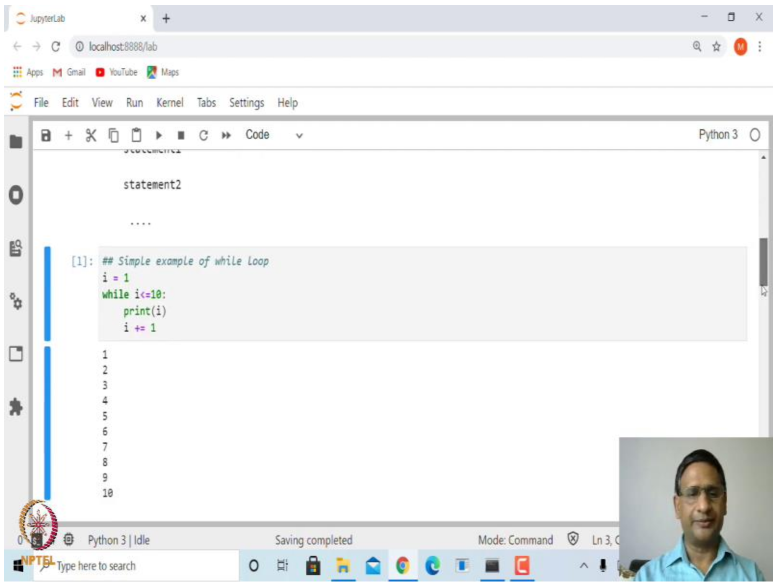
while (condition/s):

```
    statement1

    statement2
```

If you recall the for loop, actually it was used to iterate over elements of a list or a tuple, it could be a set or dictionary, and that is why in python for loops are also known as iterators. The for loop is more appropriate, when we know number of iterations in advance. Whereas, in many cases the number of iterations which is required in order to execute a particular task is not known and in that case we can use while loop. So, the while loop is preferred where the number of iterations is not specified. Of course, in case of for loop where we saw that in case the number of iteration is not known, we can break the iteration once certain condition is satisfied.

So, now let us look at how to use while loop. So, generic syntax for while loop is given by this. While, you can write inside the bracket or without bracket one condition or more than one conditions, and then you execute statement1 statement2 and so on. So, this is the block of codes that will get executed as long as this condition is satisfied.

(Refer Slide Time: 02:30)



```
    statement2

    ....
```

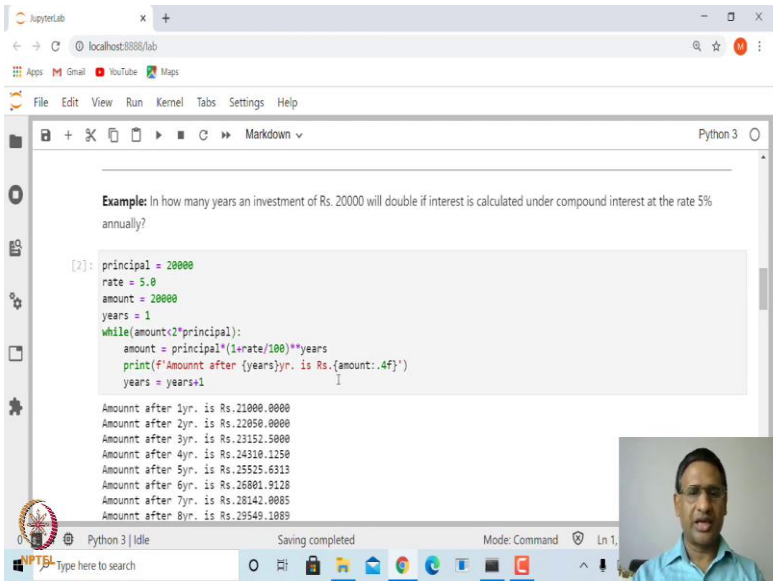
```
[1]: ## Simple example of while Loop
     i = 1
     while i<=10:
         print(i)
         i += 1
```

```
1
2
3
4
5
6
7
8
9
10
```

Let us look at a very simple example. Suppose we want to print 1 to 10 using a while loop, so how we are going to do? We will say while i is less than equal to 10, print i and increment i by 1 and you need to also initialize i equal to 1. So, how will this loop run? So, first it will check i, since i is equal to 1, which is less than equal to 10, it will go to the next, that means, it will print 1 and then i will become i plus 1, i plus equal to 1 is same as i is equal to i plus 1.

Now, i will become 2 and then since 2 is less than equal to 10, it will print 2, and then i will become 3 and so on. So, as long as i is less than equal to 10, it will print i, the moment i becomes more than 10, that is, 11 then this loop will stop. Let us execute this and you can see here you will get 1 to up to 10. In case we want to have more conditions, then we can put more conditions combined by or and. That we will see it later.

(Refer Slide Time: 03:55)



Example: In how many years an investment of Rs. 20000 will double if interest is calculated under compound interest at the rate 5% annually?

```
[2]: principal = 20000
     rate = 5.0
     amount = 20000
     years = 1
     while(amount<2*principal):
         amount = principal*(1+rate/100)**years
         print(f'Amounnt after {years}yr. is Rs.{amount:.4f}')
         years = years+1
```

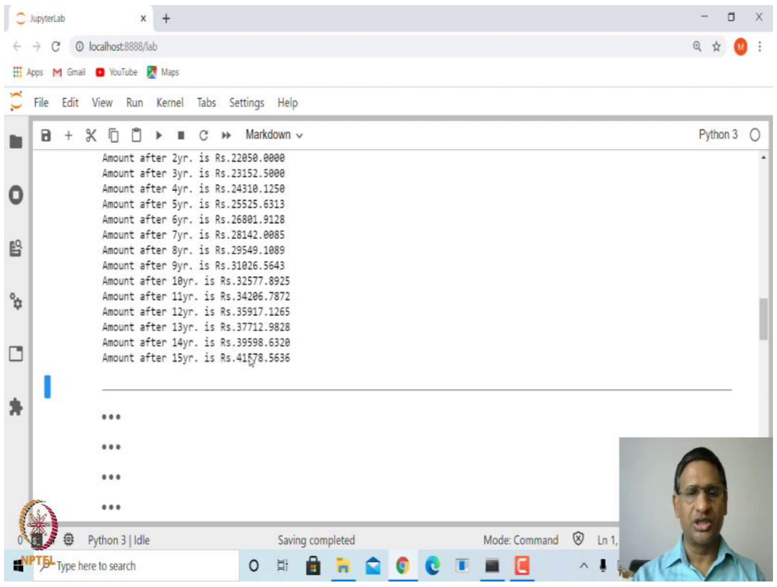
```
Amounnt after 1yr. is Rs.21000.0000
Amounnt after 2yr. is Rs.22050.0000
Amounnt after 3yr. is Rs.23152.5000
Amounnt after 4yr. is Rs.24310.1250
Amounnt after 5yr. is Rs.25525.6313
Amounnt after 6yr. is Rs.26801.9128
Amounnt after 7yr. is Rs.28142.0085
Amounnt after 8yr. is Rs.29549.1089
```

Now, let us look at an example. The example is to find the number of years for an investment of 20000 to become double using compound interest rate at 5 percent calculated annually. This we have seen using for loop, and in this case we used actually break statement when the investment became double. But natural thing is to use while loop; and how do we use while loop?

First you define the principal amount that is, 20000, interest rate 5 percent and you calculate the amount using this formula. This we have done many times earlier and as long as this amount is less than 2 times the principal you run this.

Of course, after you find the amount, you print the amount in those number of years. Since we are applying condition on amount this amount has to be initialized, so we have initialize amount to be the principal value, it will work for any initial value as long as it is less than 2 times the principal.

(Refer Slide Time: 05:21)

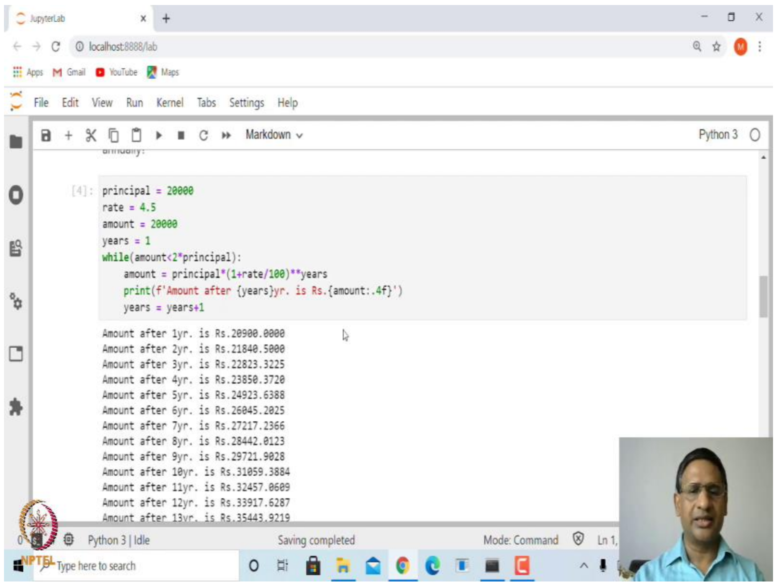


```
Amount after 2yr. is Rs.22050.0000
Amount after 3yr. is Rs.23152.5000
Amount after 4yr. is Rs.24310.1250
Amount after 5yr. is Rs.25525.6313
Amount after 6yr. is Rs.26801.9128
Amount after 7yr. is Rs.28142.0085
Amount after 8yr. is Rs.29549.1089
Amount after 9yr. is Rs.31026.5643
Amount after 10yr. is Rs.32577.8925
Amount after 11yr. is Rs.34206.7872
Amount after 12yr. is Rs.35917.1265
Amount after 13yr. is Rs.37712.9828
Amount after 14yr. is Rs.39598.6320
Amount after 15yr. is Rs.41578.5636
```

You need to increment the number of years, so let us run this. Then you run this you will get after 1 year, amount is this, I think there is spelling mistake in the amount. The amount after 1 year is 21000, after 2 years is 22050 and so on after 15 years amount becomes 41000 which is just more than double of the 20000 and therefore, it has come out of this loop.

So, in this case you see that we do not know how many years it is going to require in order to become the investment double and that is why we are using for loop.

(Refer Slide Time: 06:03)

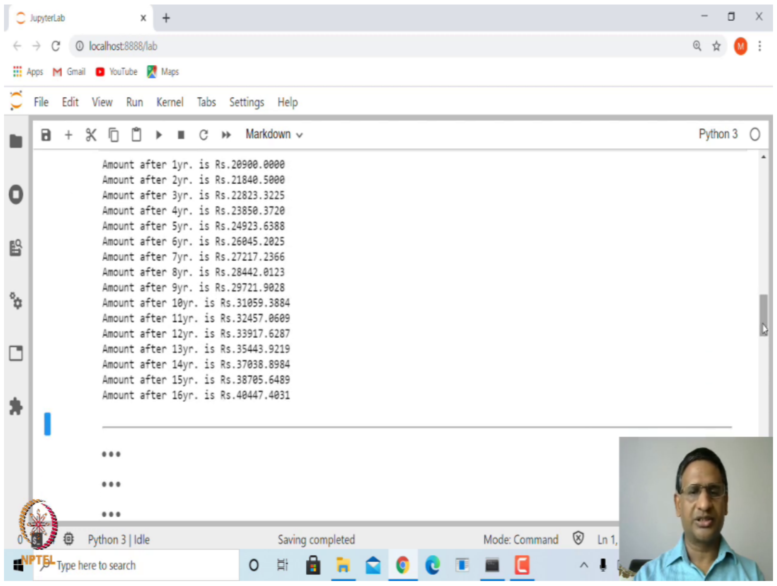


```
[4]: principal = 20000
     rate = 4.5
     amount = 20000
     years = 1
     while(amount<2*principal):
         amount = principal*(1+rate/100)**years
         print(f'Amount after {years}yr. is Rs.{amount:.4f}')
         years = years+1
```

```
Amount after 1yr. is Rs.20900.0000
Amount after 2yr. is Rs.21840.5000
Amount after 3yr. is Rs.22823.3225
Amount after 4yr. is Rs.23850.3720
Amount after 5yr. is Rs.24923.6388
Amount after 6yr. is Rs.26045.2025
Amount after 7yr. is Rs.27217.2366
Amount after 8yr. is Rs.28442.0123
Amount after 9yr. is Rs.29721.9028
Amount after 10yr. is Rs.31059.3884
Amount after 11yr. is Rs.32457.0609
Amount after 12yr. is Rs.33917.6287
Amount after 13yr. is Rs.35443.9219
```

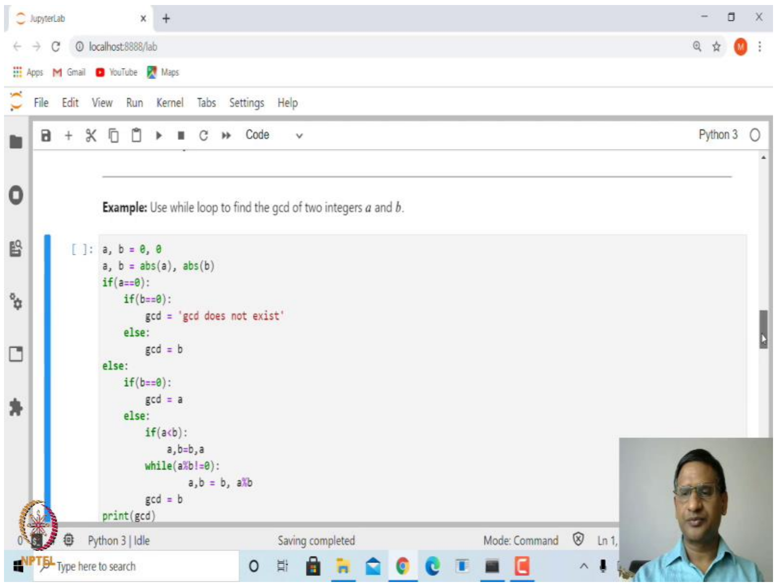
In case I reduce the interest rate for example, let me make it 4.5 percent and run this. In this case, the number of years required is 16 years, whereas, in case of 5 percentage it was 15 years.

(Refer Slide Time: 06:08)



```
Amount after 1yr. is Rs.20900.0000
Amount after 2yr. is Rs.21840.5000
Amount after 3yr. is Rs.22823.3225
Amount after 4yr. is Rs.23850.3720
Amount after 5yr. is Rs.24923.6388
Amount after 6yr. is Rs.26045.2025
Amount after 7yr. is Rs.27217.2366
Amount after 8yr. is Rs.28442.0123
Amount after 9yr. is Rs.29721.9028
Amount after 10yr. is Rs.31059.3884
Amount after 11yr. is Rs.32457.0609
Amount after 12yr. is Rs.33917.6287
Amount after 13yr. is Rs.35443.9219
Amount after 14yr. is Rs.37038.8984
Amount after 15yr. is Rs.38705.6489
Amount after 16yr. is Rs.40447.4031
```

(Refer Slide Time: 06:20)



Example: Use while loop to find the gcd of two integers *a* and *b*.

```
[ ]: a, b = 0, 0
     a, b = abs(a), abs(b)
     if(a==0):
         if(b==0):
             gcd = 'gcd does not exist'
         else:
             gcd = b
     else:
         if(b==0):
             gcd = a
         else:
             if(a<b):
                 a,b=b,a
             while(a%b!=0):
                     a,b = b, a%b
             gcd = b
     print(gcd)
```

Let us look at another example. The next example is to find gcd of two integers a and b. GCD stands for greatest common divisor. So, if you have two integers, find all the divisors of a and b, positive divisors actually, and find the one which is the greatest among the two divisors, the divisors of a and divisors of b. That is what is called greatest common divisor.

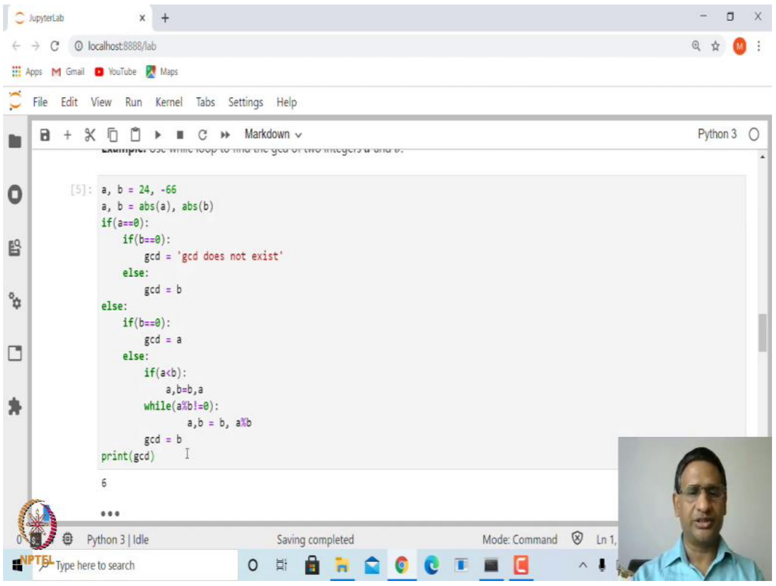
Now, in case one of them let us say a is 0 and b is nonzero, then the greatest common divisor will be b. In case b is 0, a is nonzero, the greatest common divisor will be a. In case both are 0, a and b both are 0, then any integer divides 0 and therefore, there will not be any greatest common divisor. So, you can say that greatest common divisor does not exist in this case.

There are different ways of finding greatest common divisors, one can list all the divisors or you can use division algorithm. That means, let us assume that a is greater than b, then you divide a by b and in case the remainder is 0, b is the greatest common divisor, provided b is positive.

First of all we will use that greatest common divisor of a and b is same as greatest common divisor of mod of a and mod of b. It is better, you can take a and b to be positive, in case they are negative you can convert into its modulus.

In case b does not divide a, you will get remainder, suppose the remainder is r, then next time you divide b by r, and then look at, in case the remainder is 0 the gcd will become r. You continue this process until the remainder becomes 0. This is one algorithm to find gcd of two integers. This is known as division algorithm. So, let us see how we can make use of division algorithm in order to write program for finding gcd.

(Refer Slide Time: 08:31)



```
[5]: a, b = 24, -66
a, b = abs(a), abs(b)
if(a==0):
    if(b==0):
        gcd = 'gcd does not exist'
    else:
        gcd = b
else:
    if(b==0):
        gcd = a
    else:
        if(a<b):
            a,b=b,a
        while(a%b!=0):
                a,b = b, a%b
        gcd = b
print(gcd)
```

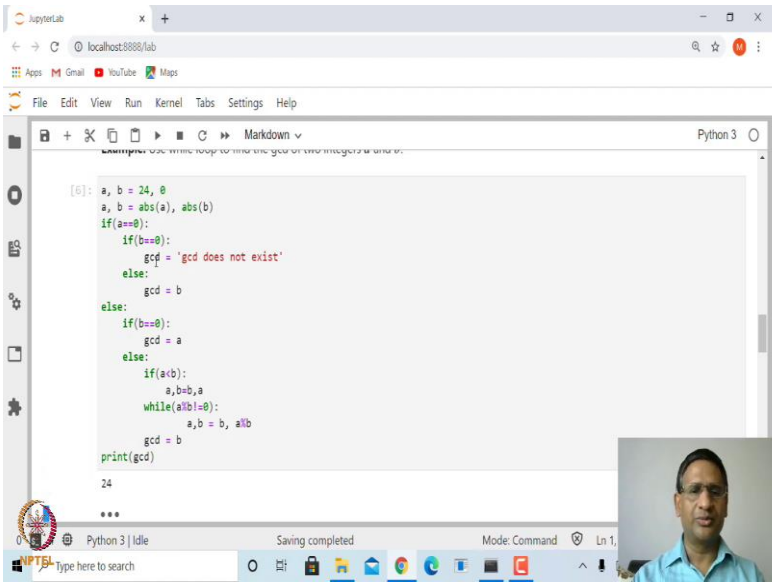
```
6
```

This is how we can write. First we have, let us say a is 24 and b is let us say 66. And a and b will take the absolute value of a and the absolute value of b, because it is possible that one of them is negative or both of them are negative.

For example, let us say b is minus 66. Convert a and b to absolute value. In case a is 0 then check whether b is 0, if b is 0 then a and b both are 0, in that case gcd does not exist. In case b is nonzero, then gcd is b. Similarly, if a is nonzero then check b. If b is 0 then gcd will be a, otherwise it means, both a and b are nonzero, in that case we will first check whether a is bigger than b. In case a is less than b, you interchange a and b and then apply division algorithm.

So, division algorithm means, divide a by b, and in case the remainder is 0 you find out the remainder, that is a by b and store that in b and then a will take value of b. And then once the remainder is 0 then it will come out of this loop and then you print what is the last value, that is the last value here b and that is b is nonzero and since a and b both are positive, you will get the gcd. Let us execute this. In this case gcd of 24 and minus 66 is 6.

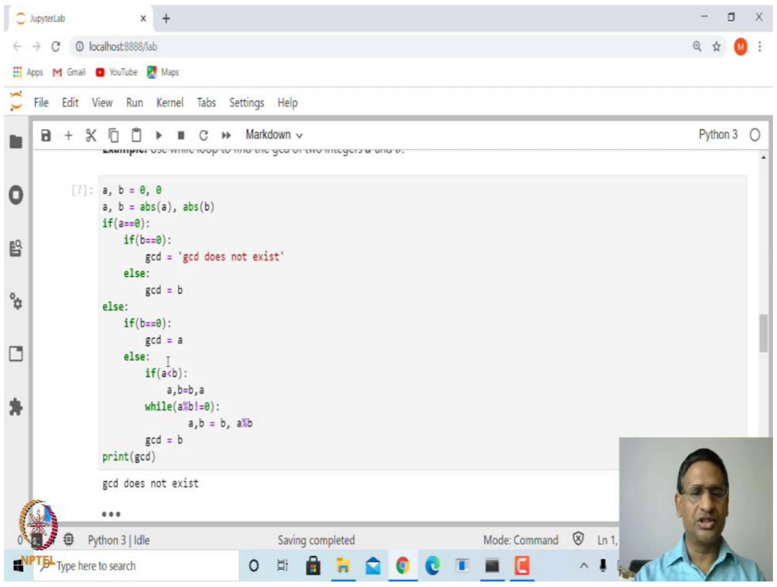
(Refer Slide Time: 10:10)



```
[6]: a, b = 24, 0
     a, b = abs(a), abs(b)
     if(a==0):
         if(b==0):
             gcd = 'gcd does not exist'
         else:
             gcd = b
     else:
         if(b==0):
             gcd = a
         else:
             if(a<b):
                 a,b=b,a
             while(a%b!=0):
                     a,b = b, a%b
             gcd = b
     print(gcd)

     24
```

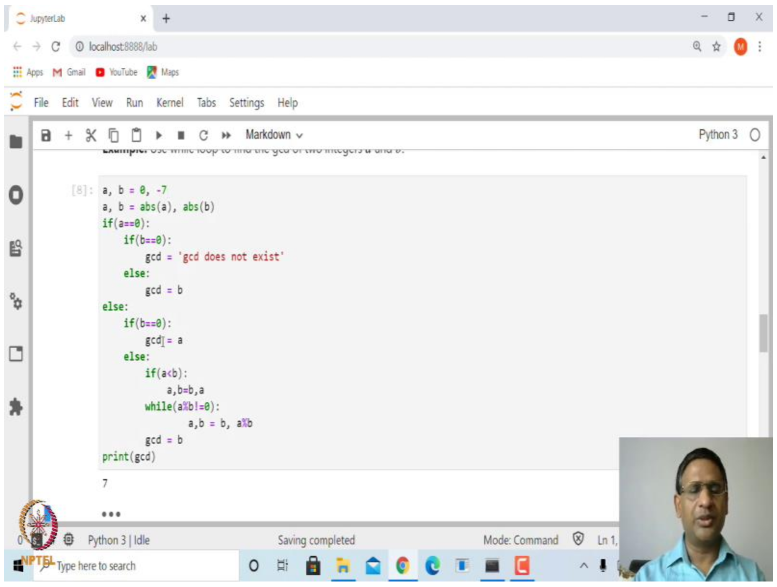
For example, if I put b equal to 0, gcd will become 24.

(Refer Slide Time: 10:13)


```
[7]: a, b = 0, 0
     a, b = abs(a), abs(b)
     if(a==0):
         if(b==0):
             gcd = 'gcd does not exist'
         else:
             gcd = b
     else:
         if(b==0):
             gcd = a
         else:
             if(a<b):
                 a,b=b,a
             while(a%b!=0):
                     a,b = b, a%b
             gcd = b
     print(gcd)

     gcd does not exist
```

If I put a and b both 0, gcds will not exist.

(Refer Slide Time: 10:22)

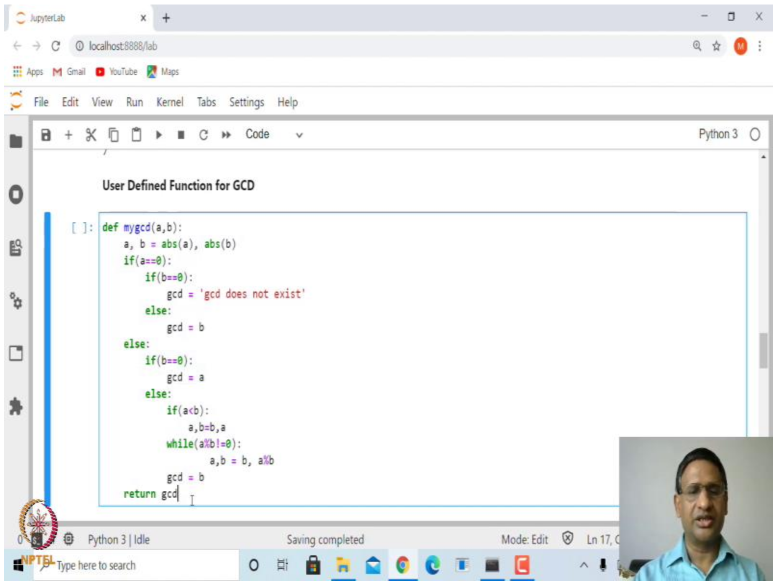

```
[8]: a, b = 0, -7
     a, b = abs(a), abs(b)
     if(a==0):
         if(b==0):
             gcd = 'gcd does not exist'
         else:
             gcd = b
     else:
         if(b==0):
             gcd = a
         else:
             if(a<b):
                 a,b=b,a
             while(a%b!=0):
                     a,b = b, a%b
             gcd = b
     print(gcd)

     7
```

If I have a is equal to 0 and b is equal, let us say minus 7, then gcd will become 7.

Now, suppose we want to define user defined function for gcd. We have already seen there is a user defined function, gcd in the math module.

But suppose we want to create our own user defined function, let me call this as mygcd. So, then in that case if you input a and b two integers a and b then it should give me the output. So, in this case what you can do is, you can simply copy paste this inside the body of the function and of course, you need to indent this. And instead of print gcd, I will say return gcd, So, let us see.

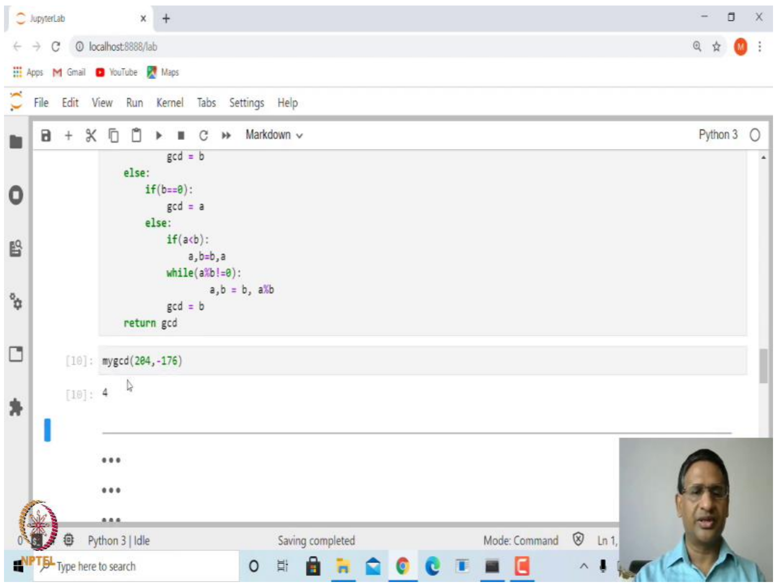
(Refer Slide Time: 11:05)


User Defined Function for GCD

```
[ ]: def mygcd(a,b):
         a, b = abs(a), abs(b)
         if(a==0):
             if(b==0):
                 gcd = 'gcd does not exist'
             else:
                 gcd = b
         else:
             if(b==0):
                 gcd = a
             else:
                 if(a<b):
                     a,b=b,a
                 while(a%b!=0):
                     a,b = b, a%b
                 gcd = b
         return gcd
```

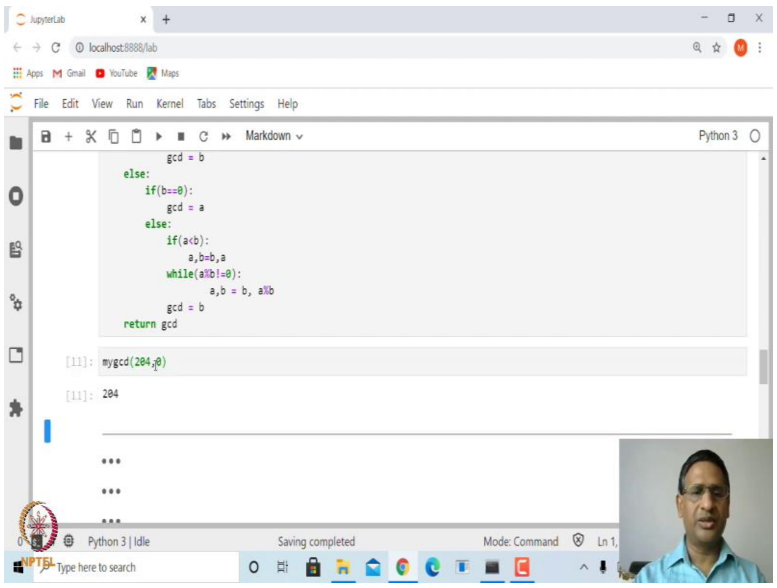
I have called this as mygcd a and b, and if you noticed I have simply copied this particular code from the above syntax and put it here and at the end I am just saying return gcd.

(Refer Slide Time: 11:25)


```
                 gcd = b
         else:
             if(b==0):
                 gcd = a
             else:
                 if(a<b):
                     a,b=b,a
                 while(a%b!=0):
                     a,b = b, a%b
                 gcd = b
         return gcd

[10]: mygcd(204,-176)

[10]: 4
```

Let us run this. Suppose we call this function mygcd of 204 and minus 176, the answer is 4.

(Refer Slide Time: 11:33)



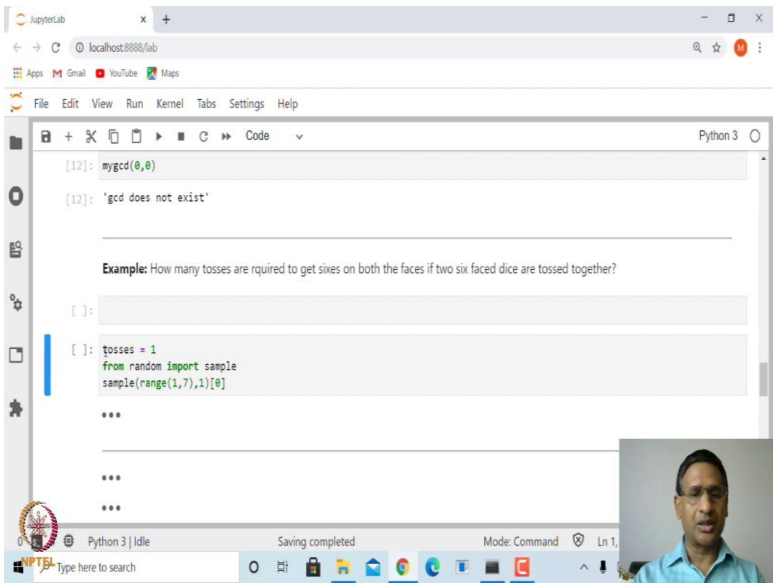
```
                gcd = b
        else:
            if(b==0):
                gcd = a
            else:
                if(a<b):
                    a,b=b,a
                while(a%b!=0):
                        a,b = b, a%b
                gcd = b
        return gcd
```

```
[11]: mygcd(204,0)
```

```
[11]: 204
```

Suppose, this is 0, then it will give me 204, if both are 0 it will say gcd does not exist. So, this is how you can make use of this division algorithm to find gcd and we have created user defined function for the same.

(Refer Slide Time: 11:38)



```
[12]: mygcd(0,0)
```

```
[12]: 'gcd does not exist'
```

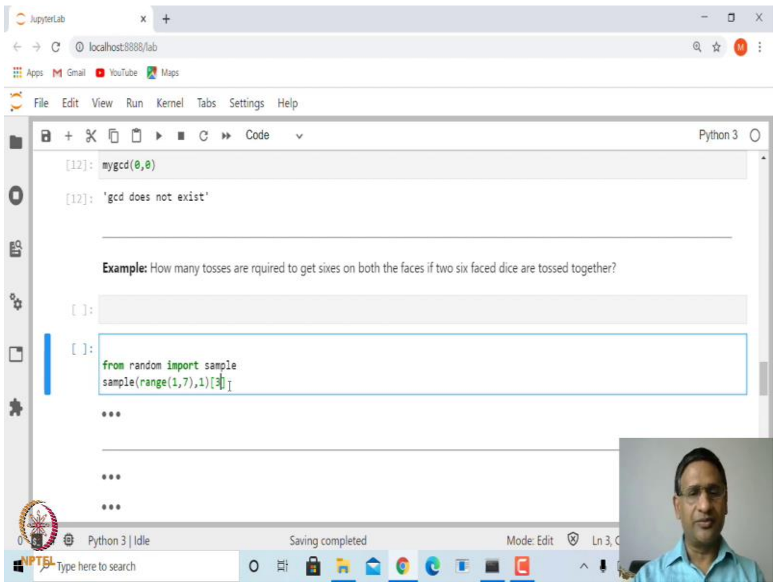
Example: How many tosses are rquired to get sixes on both the faces if two six faced dice are tossed together?

```
[ ]: tosses = 1
     from random import sample
     sample(range(1,7),1)[0]
```

Let us look at another example, the example is how many tosses are required to get sixes on both the faces, if two 6 faced dice are tossed together. So, if you are tossing two dice together and on both you want 6, how many times you need to toss in order to get one? But of course, you can see that this will vary, sometimes you can get the both the sixes in first toss itself or you may not get after so many tosses also.

Now, in order to do this problem, for tossing, we will generate a random number between 1 and 6 and that random number can be thought of as a outcome on the face of a dice when it is tossed.

(Refer Slide Time: 12:52)



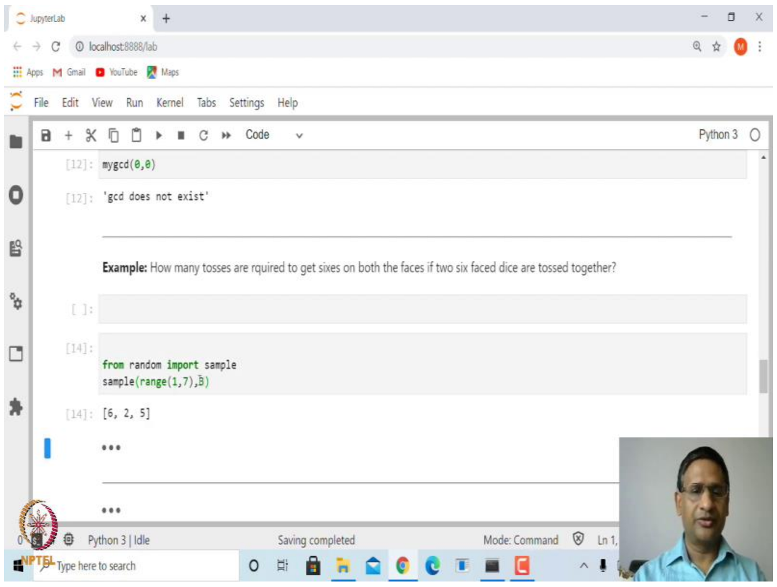
```
[12]: mygcd(0,0)

[12]: 'gcd does not exist'
```

Example: How many tosses are rquired to get sixes on both the faces if two six faced dice are tossed together?

```
[ ]:

[ ]: from random import sample
     sample(range(1,7),1)[3]
```

Let us see, there is function called, sample inside random module. So, for example, I will say import sample from random module, and then if I use sample range 1 to 7, that means, 1, 2, 3, 4, 5, up to 6 and this is the how many you want out of these numbers.

(Refer Slide Time: 13:21)



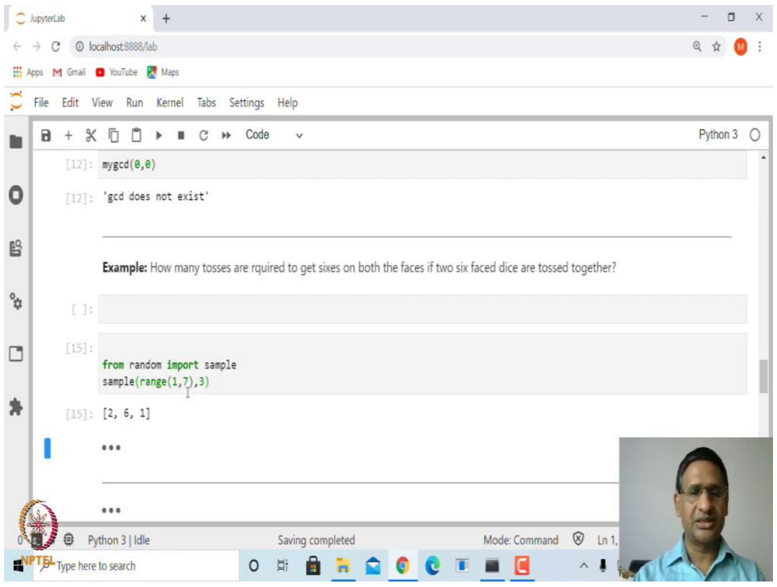
```
[12]: mygcd(0,0)

[12]: 'gcd does not exist'
```

Example: How many tosses are rquired to get sixes on both the faces if two six faced dice are tossed together?

```
[ ]:

[14]:
from random import sample
sample(range(1,7),3)

[14]: [6, 2, 5]
```

If I want to take a sample of 3 from 1 to 6, it will get, sorry it will be 3 here. So, you will get a list of 3 random numbers.

(Refer Slide Time: 13:28)



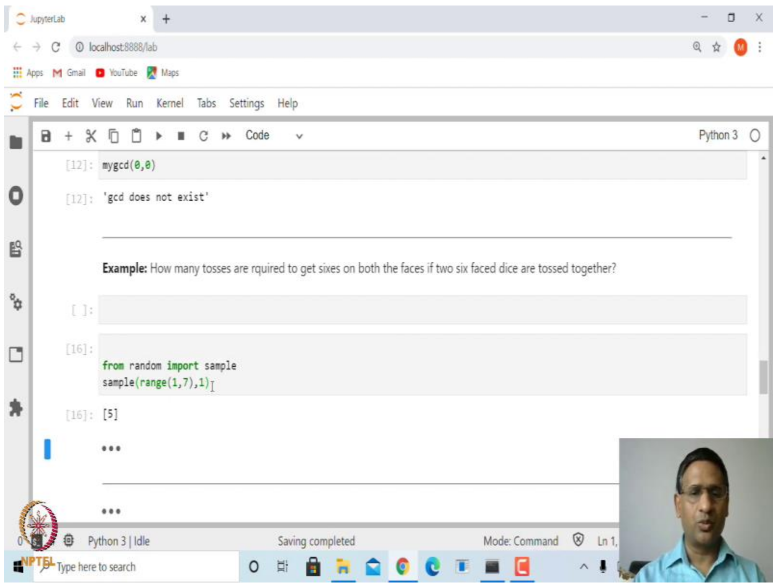
```
[12]: mygcd(0,0)

[12]: 'gcd does not exist'
```

Example: How many tosses are rquired to get sixes on both the faces if two six faced dice are tossed together?

```
[ ]:

[15]:
from random import sample
sample(range(1,7),3)

[15]: [2, 6, 1]
```

If I do once more I will get another 3 numbers.

(Refer Slide Time: 13:32)



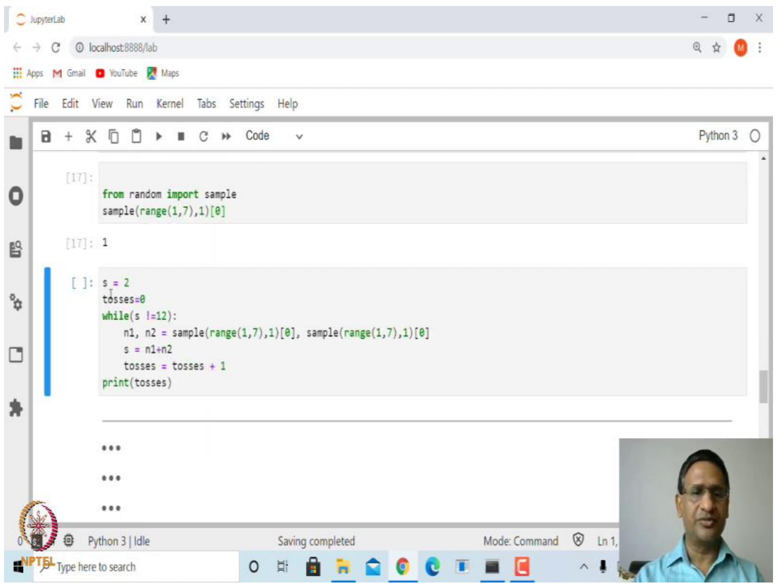
```
[12]: mygcd(0,0)

[12]: 'gcd does not exist'
```

Example: How many tosses are rquired to get sixes on both the faces if two six faced dice are tossed together?

```
[ ]:

[16]:
      from random import sample
      sample(range(1,7),1)

[16]: [5]
```

If I want to generate only one, then I will put here 1.

(Refer Slide Time: 13:41)



```
[17]:
      from random import sample
      sample(range(1,7),1)[0]

[17]: 1

[ ]: s = 2
     tosses=0
     while(s !=12):
         n1, n2 = sample(range(1,7),1)[0], sample(range(1,7),1)[0]
         s = n1+n2
         tosses = tosses + 1
     print(tosses)
```

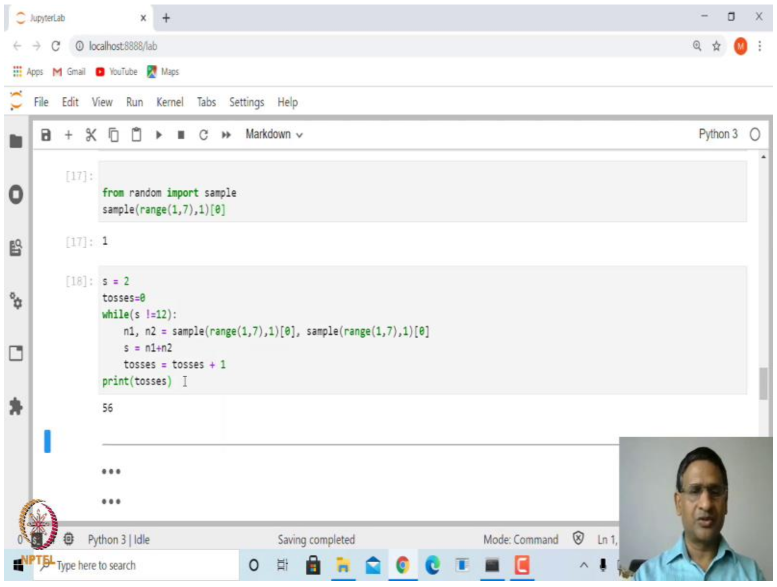
So, I will take the first element of that that is 0th index and that you will get this.

So, now we will make use of this in order to find how many tosses are required in order to get both sixes.

So, we can use again while loop. When we use while loop, first we need to initialize the number of tosses which is equal to let us say 0, and we generate two numbers n1 and n2, which is the sample from 1 to 7, one sample, and 0th element, that is, that take the number, We have done already here. So, this will generate n1 and n2 and then find S to be n1 plus n2.

Now, in case both n1, n2 are 6 the sum will become 12. So, in case sum is equal to 12 you need to come out of this, otherwise you need to run this again. As long as s is not equal to 12 you continue this process and that is why s has to be initialized. You can initialize anything less than 12, but I have initialized equal to 2, which is that is the minimum value that you can get when you toss two dice together.

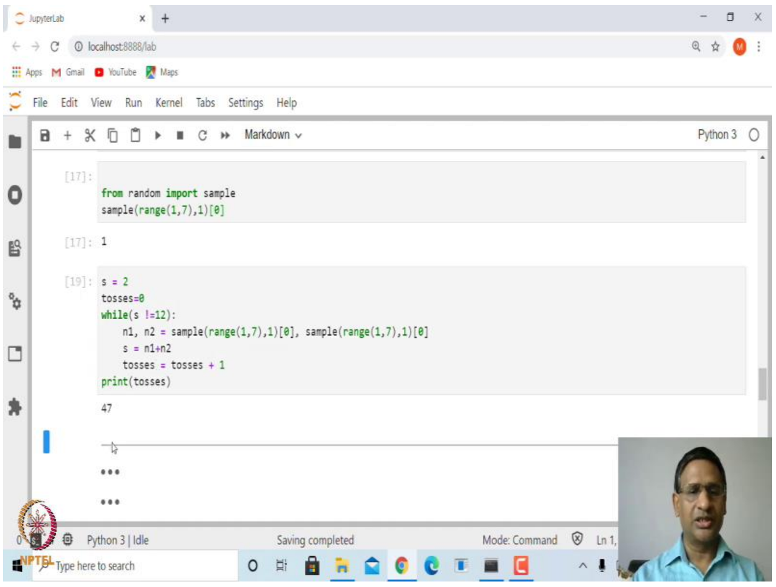
(Refer Slide Time: 14:57)



```
[17]:
      from random import sample
      sample(range(1,7),1)[0]

[17]: 1

[18]: s = 2
      tosses=0
      while(s !=12):
          n1, n2 = sample(range(1,7),1)[0], sample(range(1,7),1)[0]
          s = n1+n2
          tosses = tosses + 1
      print(tosses)

      56
```

So, let us run this. So, when you run this first time 56 tosses are required.

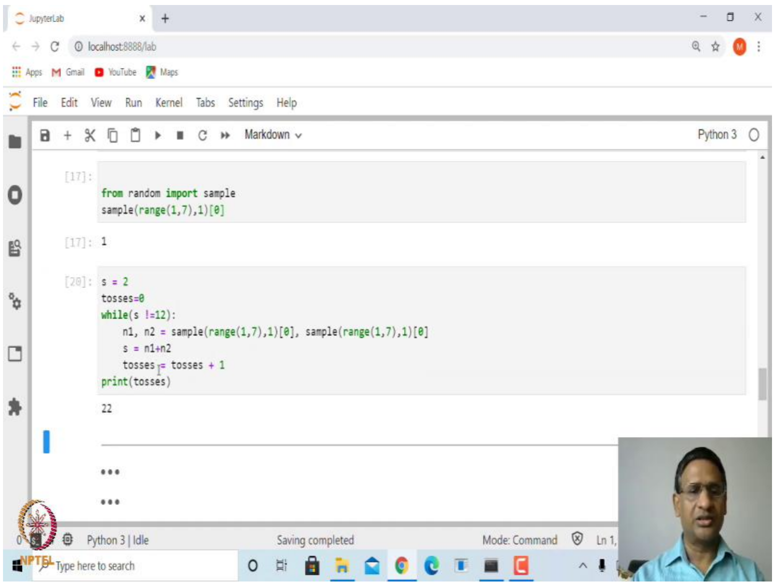
(Refer Slide Time: 15:02)



```
[17]:
from random import sample
sample(range(1,7),1)[0]

[17]: 1

[19]: s = 2
tosses=0
while(s !=12):
    n1, n2 = sample(range(1,7),1)[0], sample(range(1,7),1)[0]
    s = n1+n2
    tosses = tosses + 1
print(tosses)

47
```

I run once more, this time 47 tosses are required.

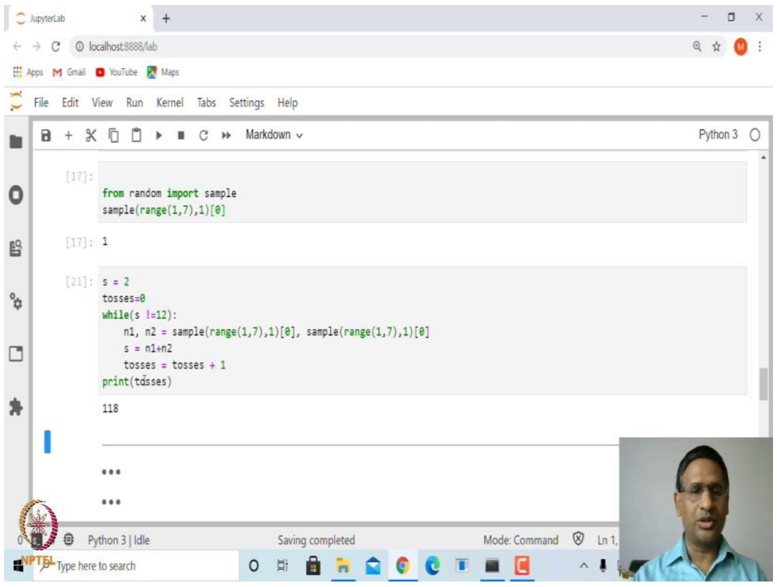
(Refer Slide Time: 15:05)



```
[17]:
from random import sample
sample(range(1,7),1)[0]

[17]: 1

[20]: s = 2
tosses=0
while(s !=12):
    n1, n2 = sample(range(1,7),1)[0], sample(range(1,7),1)[0]
    s = n1+n2
    tosses = tosses + 1
print(tosses)

22
```

If I run once more 22.

(Refer Slide Time: 15:09)

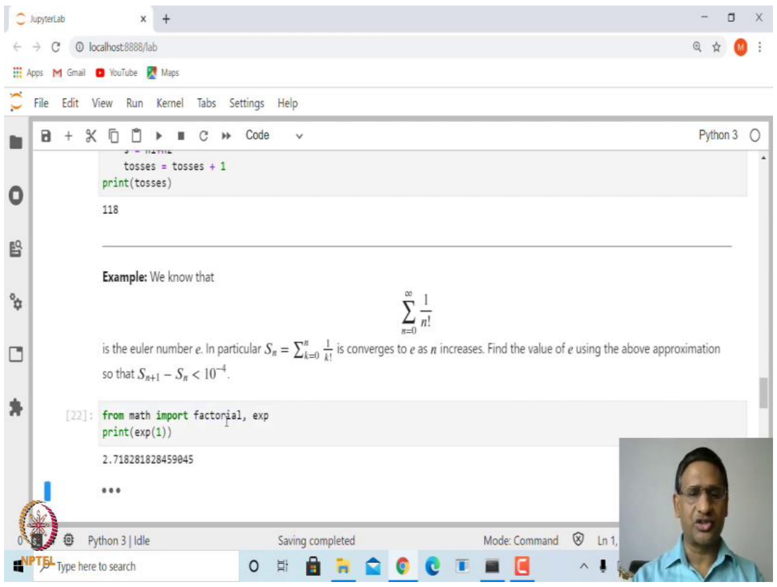


```
[17]: from random import sample
      sample(range(1,7),1)[0]

[17]: 1

[21]: s = 2
      tosses=0
      while(s !=12):
          n1, n2 = sample(range(1,7),1)[0], sample(range(1,7),1)[0]
          s = n1+n2
          tosses = tosses + 1
      print(tosses)

      118
```

If I runs one run once more it is 118.

Of course, it will vary, the number of tosses is not same because it is a random. This is a simple game that can be generated using this.

(Refer Slide Time: 15:24)



```
    tosses = tosses + 1
print(tosses)

118
```

Example: We know that

$$\sum_{n=0}^{\infty} \frac{1}{n!}$$

is the euler number e. In particular $S_n = \sum_{k=0}^{n} \frac{1}{k!}$ is converges to e as n increases. Find the value of e using the above approximation so that $S_{n+1} - S_n < 10^{-4}$.

```
[22]: from math import factorial, exp
      print(exp(1))

      2.718281828459045
```

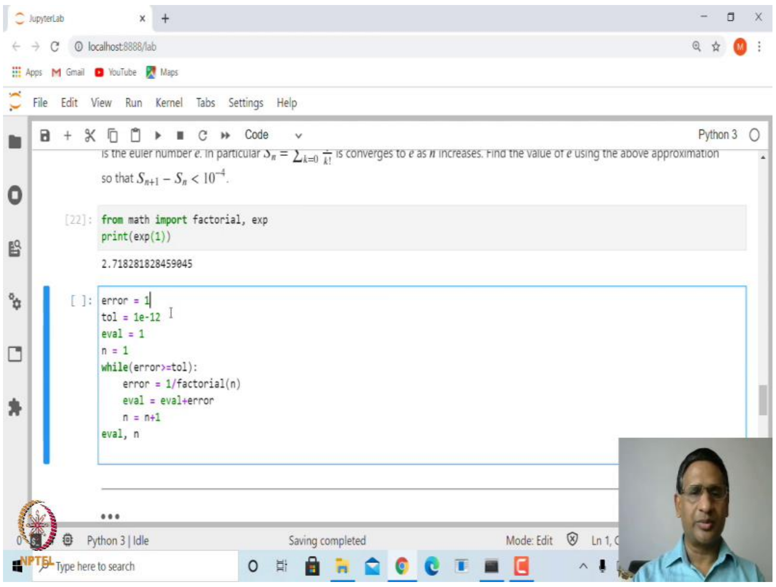
Now, let us look at another example. All of you must have seen this series summation one upon n factorial, n going from 0 to infinity. This actually converges to an Euler number which is e, which is denoted by e and its value 2.718 something. So, we want to find the value of this Euler number within certain range, which is correct up to certain decimal number of decimal places.

If you see here, we cannot find infinite sum. So, in this case what we will do is, we will define what is called partial sums Sn, and this is a sequence actually. So, Sn converges to e as n goes to infinity, that is, what you have. Also it is increasing sequence.

And if you look at Sn and Sn plus 1, the Sn plus 1 will have much better approximation than Sn. If I look at Sn plus 1 minus Sn, that is, the error term, you can think of. That error term, once it is less than 10 power minus 4, you can stop.

So, let us see how we can make use of this. First of all, let us find out what is the value of exponential 1, that is Euler number, and since we are going to make use of factorial, we know that it exist in math module. So, we will import both exponential and factorial functions. And the value of exponential of 1 that is, Euler number, is 2.7182818284 dot dot, this is up to certain number of decimal places.

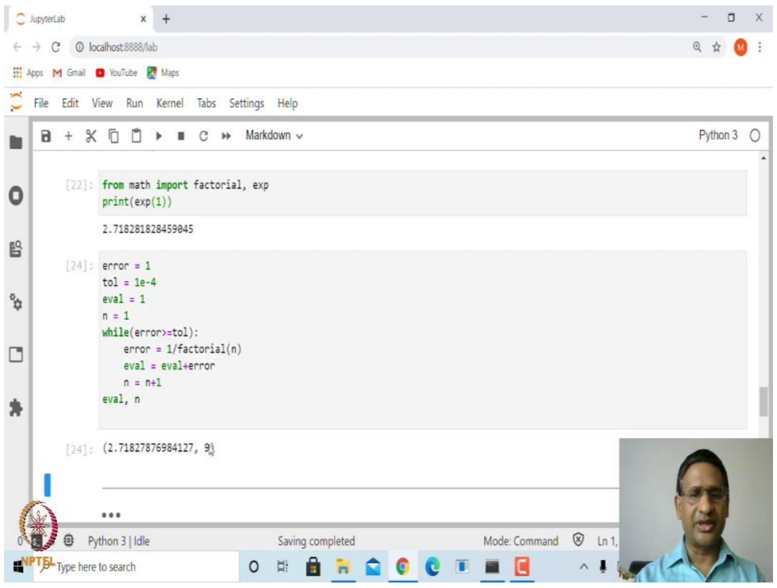
(Refer Slide Time: 17:07)


is the euler number e. In particular $S_n = \sum_{k=0} \frac{1}{k!}$ is converges to e as n increases. Find the value of e using the above approximation so that $S_{n+1} - S_n < 10^{-4}$.

```
[22]: from math import factorial, exp
      print(exp(1))

2.718281828459045

[ ]: error = 1
     tol = 1e-12
     eval = 1
     n = 1
     while(error>=tol):
         error = 1/factorial(n)
         eval = eval+error
         n = n+1
     eval, n
```

Now, let us use while loop. This is the error term we are thinking. So, this error if it is less than 10 power minus 4, we have to stop. We initialize the error as 1, and the tolerance we have, 10 to power minus 4. Let me change this to 10 to power minus 4.

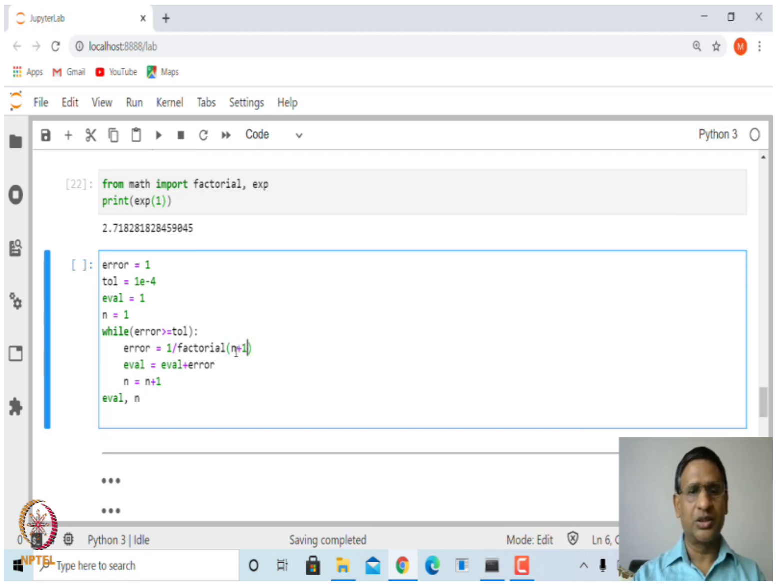
(Refer Slide Time: 17:23)



```
[22]: from math import factorial, exp
      print(exp(1))

2.718281828459045

[24]: error = 1
      tol = 1e-4
      eval = 1
      n = 1
      while(error>=tol):
          error = 1/factorial(n)
          eval = eval+error
          n = n+1
      eval, n

[24]: (2.71827876984127, 9)
```

The eval, that is the approximate value of exponential of 1. So, that again is initialized as 1. So, now initialize as 1 which is nothing but 0 factorial, 1 upon 0 factorial.

(Refer Slide Time: 17:54)

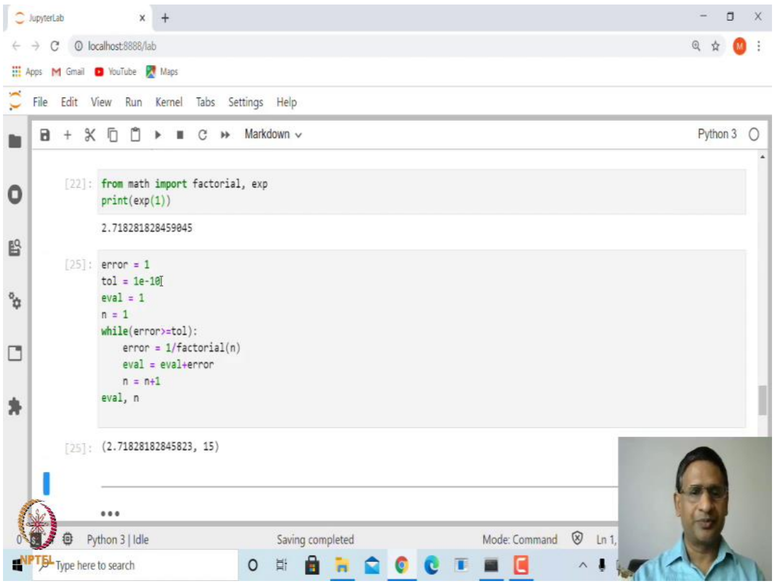


```
[22]: from math import factorial, exp
      print(exp(1))

2.718281828459045

[ ]: error = 1
     tol = 1e-4
     eval = 1
     n = 1
     while(error>=tol):
         error = 1/factorial(n+1)
         eval = eval+error
         n = n+1
     eval, n
```

So, while error is bigger than the tolerance, you find out, one upon n factorial which is actually the error, the error is 1. And since we are starting at n equal to 1, this is will be error.

So, first time 1 upon 1 factorial is 1. We started with 1, so this will become 2. Now, this error is greater than equal to tolerance, so again it will go to this place and run. Of course, you need to increment n by 1 and let us say evaluate this. If I evaluate this after 9 terms, we have got this correct up to 4 decimal places.

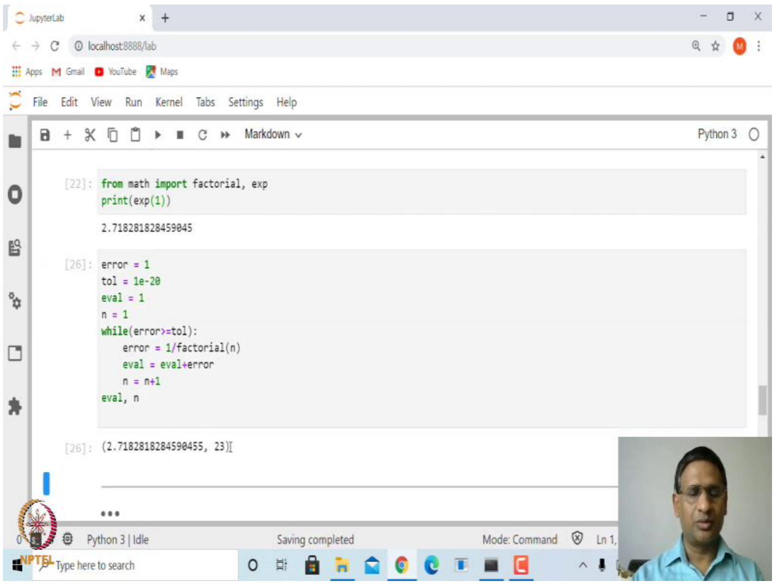
(Refer Slide Time: 18:36)



```
[22]: from math import factorial, exp
      print(exp(1))

      2.718281828459045

[25]: error = 1
      tol = 1e-10
      eval = 1
      n = 1
      while(error>=tol):
          error = 1/factorial(n)
          eval = eval+error
          n = n+1
      eval, n

[25]: (2.71828182845823, 15)
```

If I change this to next let us say, 10 and then the number of terms which are required is more, so this is 15 in this case.

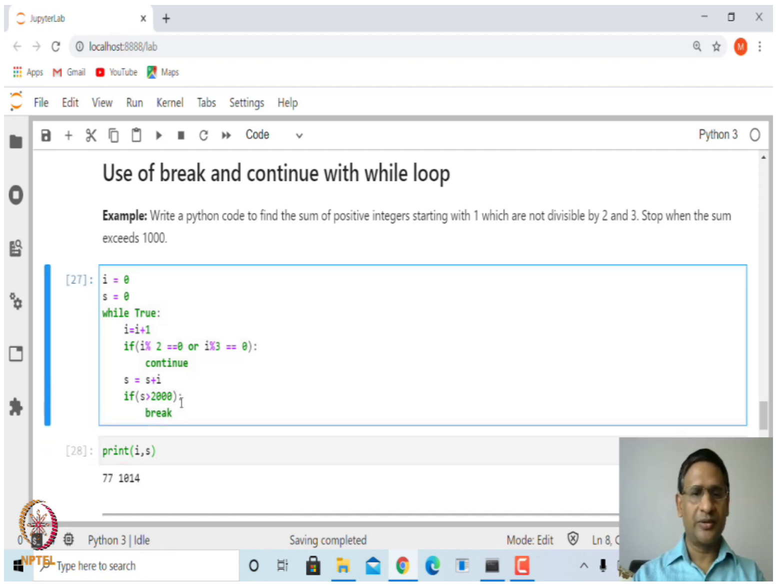
(Refer Slide Time: 18:46)



```
[22]: from math import factorial, exp
      print(exp(1))

      2.718281828459045

[26]: error = 1
      tol = 1e-20
      eval = 1
      n = 1
      while(error>=tol):
          error = 1/factorial(n)
          eval = eval+error
          n = n+1
      eval, n

[26]: (2.7182818284590455, 23)
```

If I change it to let us say 20 then it will require more number of iterates which 23. So, of course, in this case the convergence is quite fast and that is what you can see.

(Refer Slide Time: 18:59)



Use of break and continue with while loop

Example: Write a python code to find the sum of positive integers starting with 1 which are not divisible by 2 and 3. Stop when the sum exceeds 1000.

```
[27]: i = 0
      s = 0
      while True:
          i=i+1
          if(i% 2 ==0 or i%3 == 0):
              continue
          s = s+i
          if(s>2000):
              break
```

```
[28]: print(i,s)
```

```
77 1014
```

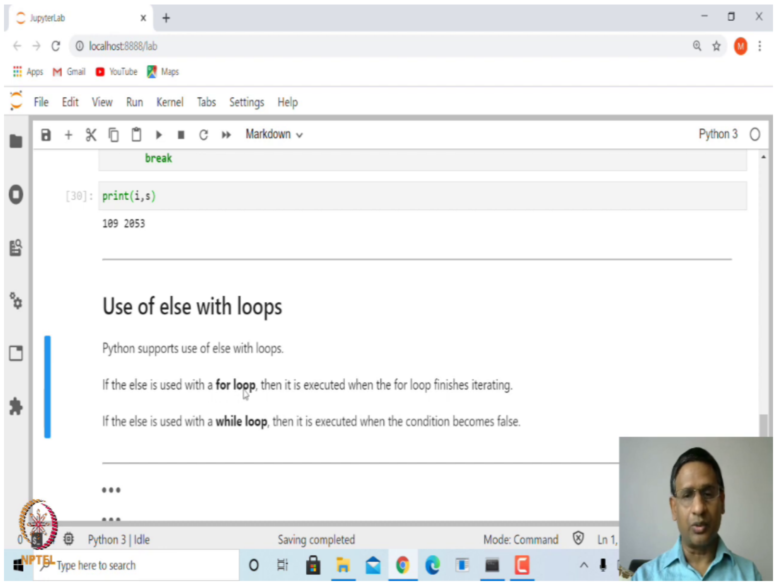
Let us look at another example. In this example now we will make use of break and continue with while loop. We have seen in the last lecture, we can make use of break and continue with for loop.

What does break do? It breaks the loop at intermediate stage in case certain condition is satisfy. What does continue do? It skips execution of that particular iteration. You can make use of break and continue along with while loop as well.

So, let us take a simple example. In this example, we want to write a python code to find the sum of positive integers starting from 1 which are not divisible by 2 and 3 both, and stop this when the sum becomes more than 1000.

So, let us see how do we make use of this? We will start with i equal to 0, the sum is equal to 0, and while true, this is actually infinite loop. So, while true, you increment i by 1. Since i equal to 0, this will become i equal to 1 and check whether it is divisible by 2 or divisible by 3. In case it is divisible by any one of these 2 or 3, then you skip because you want something which is not divisible. So, you skip that, that is use continue and then otherwise you add i to s, s was 0 in initially, so s will become s plus 1 in the first case. In case this s is more than 1000 then you break. So, in this particular code we are making use of both continue as well as break. So, let us run this. When we run this, and let us now print what should be the number of integers i such that from 1 to i the sum of the all these integers which are not divisible by 2 and 3 becomes more than 1000. This is 77. Instead of 1000, if I want let us say 2000 then you will require more iterations more number of integers, in this case it is 109.

(Refer Slide Time: 21:19)



```
        break
```

```
[30]: print(i,s)
```

```
109 2053
```

Use of else with loops

Python supports use of else with loops.

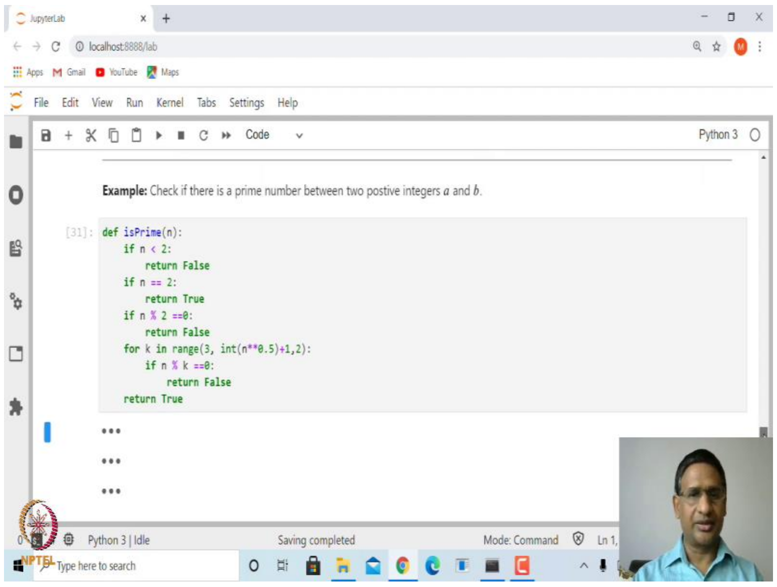
If the else is used with a **for loop**, then it is executed when the for loop finishes iterating.

If the else is used with a **while loop**, then it is executed when the condition becomes false.

Now, we can also make use of what is known as else along with any loop, for loop and while loop inside python. This is a very nice feature. Many times when none of the condition inside for loop or while loop are satisfied then you can make use of else, then you display some condition.

So, what does it say? In case, else is used with for loop then it is executed when for loop finishes iterating. So, if all the iteration are over and then only this will get executed. Similarly, in case of while loop, if you use else with while loop then it gets executed when the condition becomes false. So, if all the conditions inside the while loop are false then it will execute.

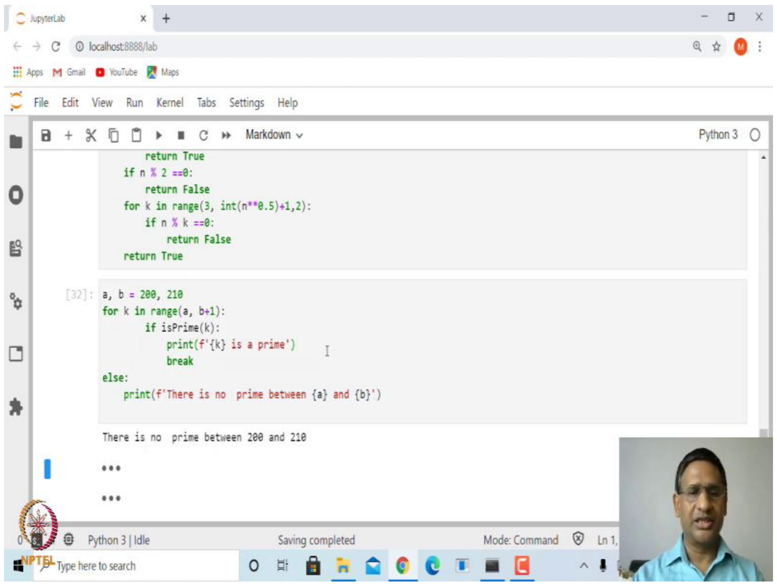
(Refer Slide Time: 22:26)


Example: Check if there is a prime number between two postive integers *a* and *b*.

```
[31]: def isPrime(n):
          if n < 2:
              return False
          if n == 2:
              return True
          if n % 2 ==0:
              return False
          for k in range(3, int(n**0.5)+1,2):
              if n % k ==0:
                  return False
          return True
```

So, let us use this. Let us look at an example of finding a prime between two integers. In case there is no prime, then you print that that the prime does not exist between the two integers a and b. So, we will make use of isPrime function which we created in the last lecture.

(Refer Slide Time: 22:47)

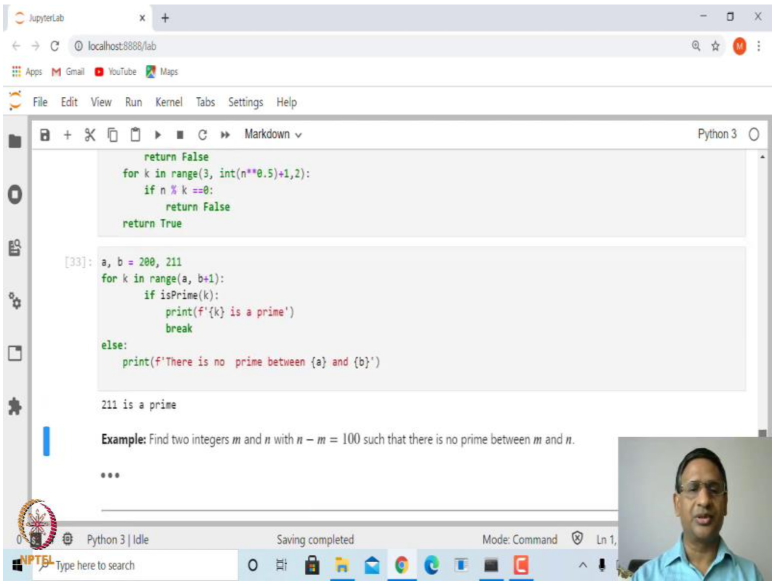

```
              return True
          if n % 2 ==0:
              return False
          for k in range(3, int(n**0.5)+1,2):
              if n % k ==0:
                  return False
          return True
```

```
[32]: a, b = 200, 210
      for k in range(a, b+1):
              if isPrime(k):
                  print(f'{k} is a prime')
                  break
      else:
          print(f'There is no  prime between {a} and {b}')
```

```
There is no  prime between 200 and 210
```

Now what do we do? We have a is equal to 200 and b is equal to, let us say 210, and for k between a and b, b included, check whether the number k is prime. In case it is a prime then you print it is a prime number and you break, otherwise if you cannot find prime then you use else and then print that that there is no prime between a and b. So, if I execute this it says that the prime between 200 and 210 does not exist. That means, every number between 200 and 210 are composite number.

(Refer Slide Time: 23:27)



```
        return False
    for k in range(3, int(n**0.5)+1,2):
        if n % k ==0:
            return False
    return True
```

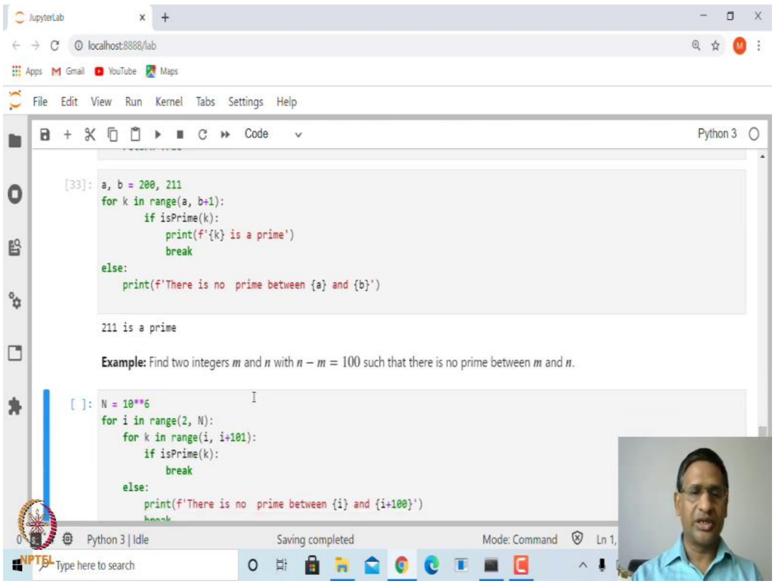
```
[33]: a, b = 200, 211
      for k in range(a, b+1):
              if isPrime(k):
                  print(f'{k} is a prime')
                  break
      else:
          print(f'There is no  prime between {a} and {b}')
```

```
211 is a prime
```

Example: Find two integers m and n with $n - m = 100$ such that there is no prime between m and n.

Suppose, instead of 210, if I make 211, then 211 is actually a prime number. Suppose you want to increase that length, instead of 210, we want to make it 100. So, between two consecutive 100 integers if there is no prime you want to find such range.

(Refer Slide Time: 23:42)



```
[33]: a, b = 200, 211
      for k in range(a, b+1):
              if isPrime(k):
                  print(f'{k} is a prime')
                  break
      else:
          print(f'There is no  prime between {a} and {b}')
```

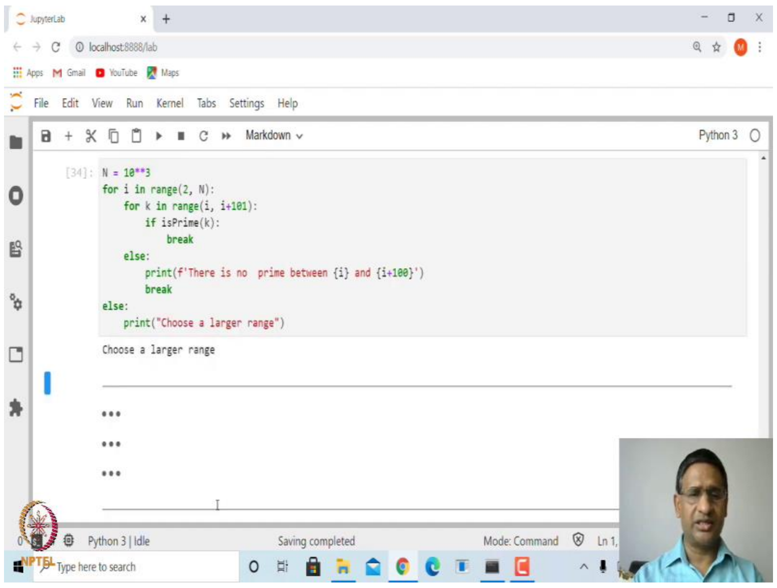
```
211 is a prime
```

Example: Find two integers m and n with $n - m = 100$ such that there is no prime between m and n.

```
[ ]: N = 10**6
     for i in range(2, N):
         for k in range(i, i+101):
             if isPrime(k):
                 break
         else:
             print(f'There is no  prime between {i} and {i+100}')
```

So, let us see. We start with let us say, we want 100.

(Refer Slide Time: 23:53)



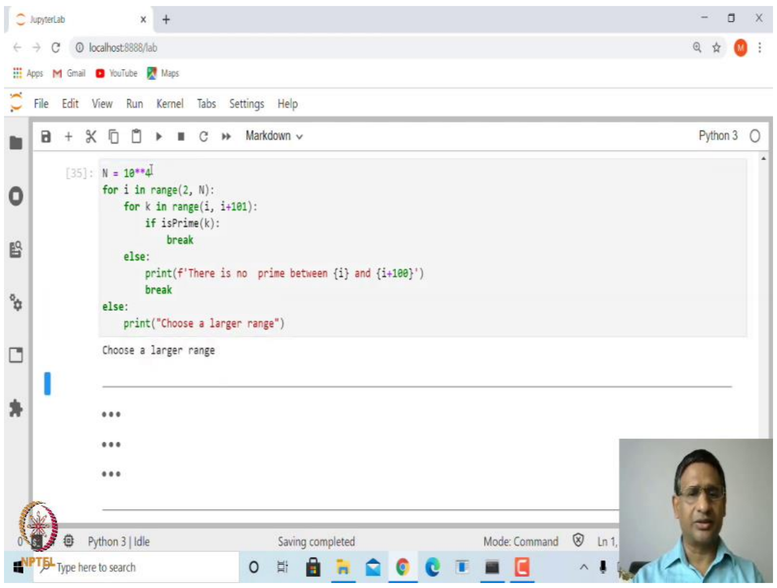
```
[34]: N = 10**3
      for i in range(2, N):
          for k in range(i, i+101):
              if isPrime(k):
                  break
          else:
              print(f'There is no  prime between {i} and {i+100}')
              break
      else:
          print("Choose a larger range")

      Choose a larger range
```

We know that there many primes between 1 and 100, so 100 is not going to work. So, let us try with 1000. So, between 1 and 1000, is there some range, let us say k and k plus 100 such that there is there are no primes between these two. So, how do we do that?

We will use nested for loop. So, first this is the loop which is just very similar to what we have here, in this case instead of a and b we are using i and i plus 101.

Then you do from 1 to 1000, 1 to 1000, so it will with 1000, then then see. If this does not happen, if it is unable to find any interval of length 100, that is, consecutive 100 integers which does not have prime number, then it will show that is 1000 is not enough. You   give a larger range. So, let us run this.

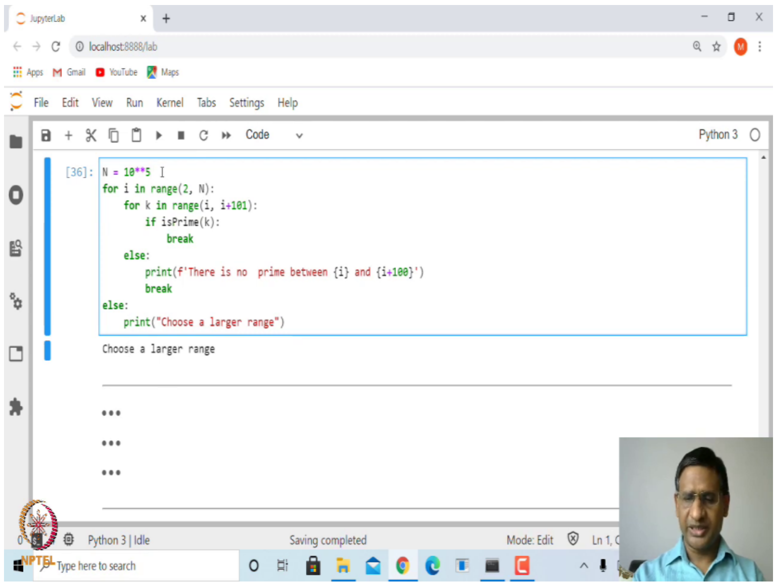
(Refer Slide Time: 25:25)



```
[35]: N = 10**4
      for i in range(2, N):
          for k in range(i, i+101):
              if isPrime(k):
                  break
          else:
              print(f'There is no  prime between {i} and {i+100}')
              break
      else:
          print("Choose a larger range")

      Choose a larger range
```

So, it says that you choose a larger range. Now instead of 1000, let us make it to the power 4 that is 10000. Still, it is unable to find.

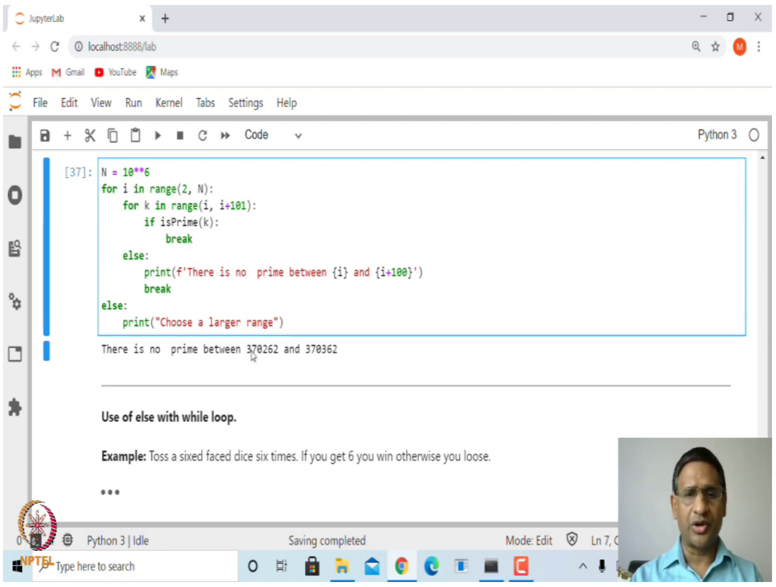
(Refer Slide Time: 25:31)



```
[36]: N = 10**5
      for i in range(2, N):
          for k in range(i, i+101):
              if isPrime(k):
                  break
          else:
              print(f'There is no  prime between {i} and {i+100}')
              break
      else:
          print("Choose a larger range")

      Choose a larger range
```

Let us make it, to the power of 5 and then it takes little while and it is still not able to find. Let us make power 6, it should be able to find in this case, but it will take more time to run, because there are so many iterations it has to run.

(Refer Slide Time: 25:44)



```
[37]: N = 10**6
      for i in range(2, N):
          for k in range(i, i+101):
              if isPrime(k):
                  break
          else:
              print(f'There is no  prime between {i} and {i+100}')
              break
      else:
          print("Choose a larger range")

      There is no  prime between 370262 and 370362
```

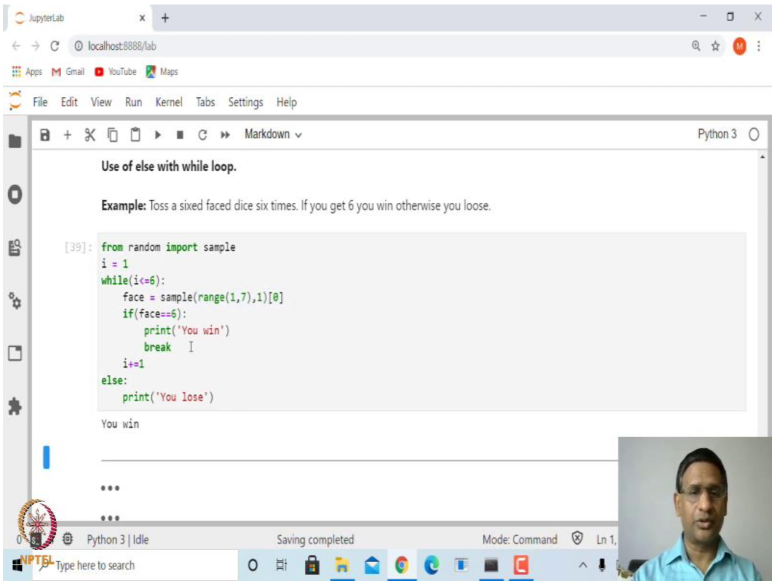
Use of else with while loop.

Example: Toss a sixed faced dice six times. If you get 6 you win otherwise you loose.

Of course, you can make this procedure faster. It is still running. Yes now it has found. So, you can it has found that 370262 and 37036,   between this range you cannot find any prime.

Of course, if they want the length not 100, but 1000 then it will require even larger range.

(Refer Slide Time: 26:31)



Use of else with while loop.

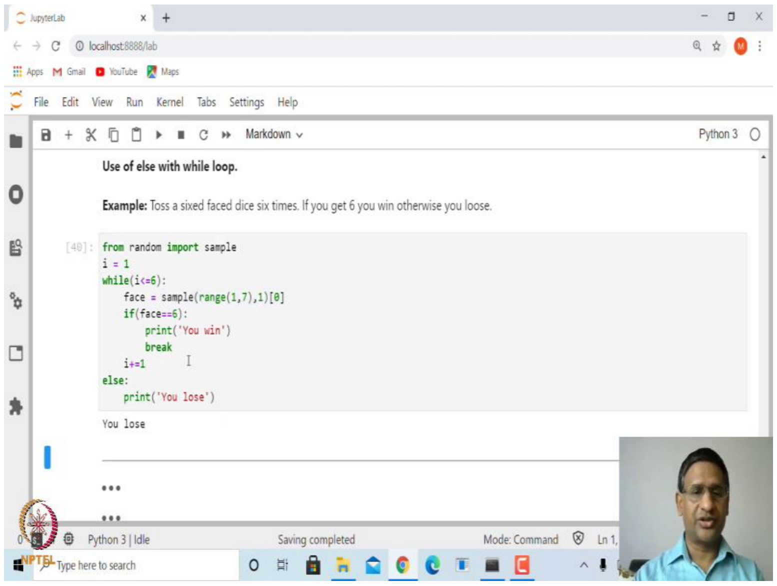
Example: Toss a sixed faced dice six times. If you get 6 you win otherwise you loose.

```
[39]: from random import sample
      i = 1
      while(i<=6):
          face = sample(range(1,7),1)[0]
          if(face==6):
              print('You win')
              break
          i+=1
      else:
          print('You lose')

      You win
```

Similarly, you can make use of else with while loop also. Suppose, you want to toss a 6 faced dice 6 times and in case you get 6 you say you win otherwise you will lose. This is a very simple thing. So, how we have made use of else here?

So, you started with initially the toss is equal to 1 and in case this toss is less than equal to 6, keep on tossing and in case the face value is equal to 6, then you print you win and break, otherwise you increment i by 1. And in case it is unable to find within this 6 tosses, any 6 then you say you lose. So, again we are making use of sample function, taking one sample from 1 to 6. So, you win.

(Refer Slide Time: 27:21)


Use of else with while loop.

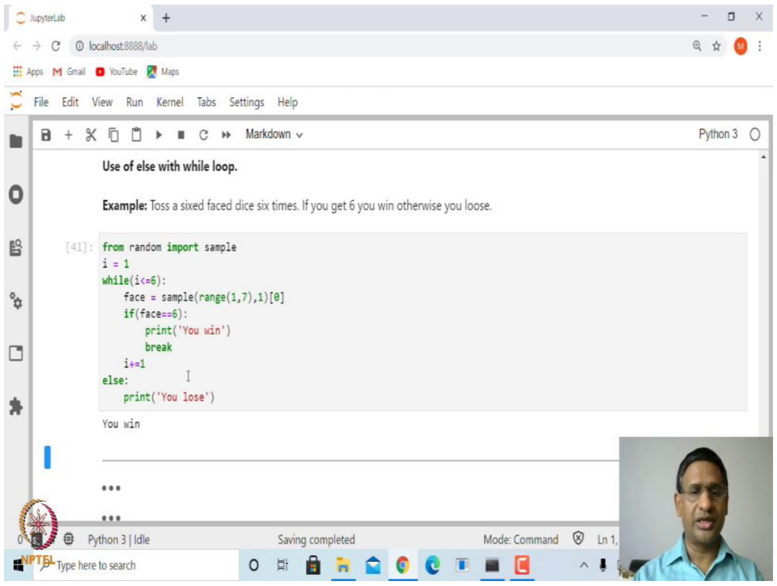
Example: Toss a sixed faced dice six times. If you get 6 you win otherwise you loose.

```
[40]: from random import sample
      i = 1
      while(i<=6):
          face = sample(range(1,7),1)[0]
          if(face==6):
              print('You win')
              break
          i+=1
      else:
          print('You lose')
```

```
You lose
```

If I say again time you lose.

(Refer Slide Time: 27:22)


Use of else with while loop.

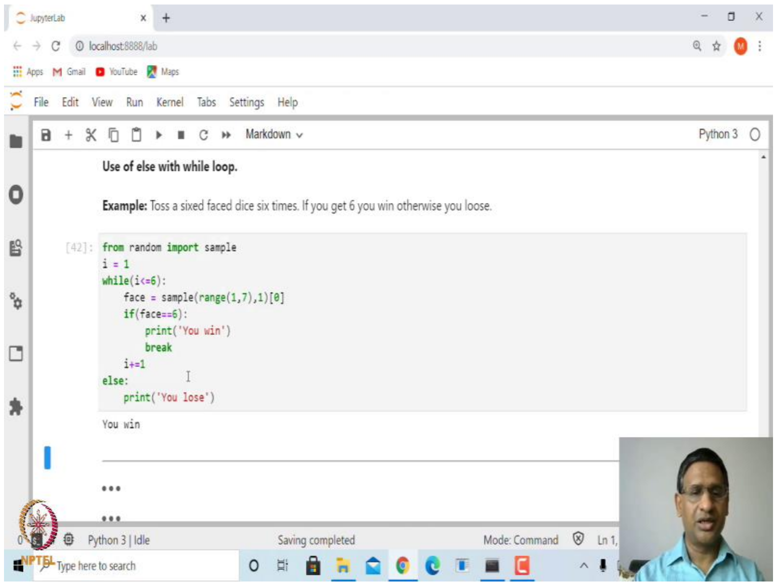
Example: Toss a sixed faced dice six times. If you get 6 you win otherwise you loose.

```
[41]: from random import sample
      i = 1
      while(i<=6):
          face = sample(range(1,7),1)[0]
          if(face==6):
              print('You win')
              break
          i+=1
      else:
          print('You lose')
```

```
You win
```

If I say again, this time you lose.

(Refer Slide Time: 27:24)



Use of else with while loop.

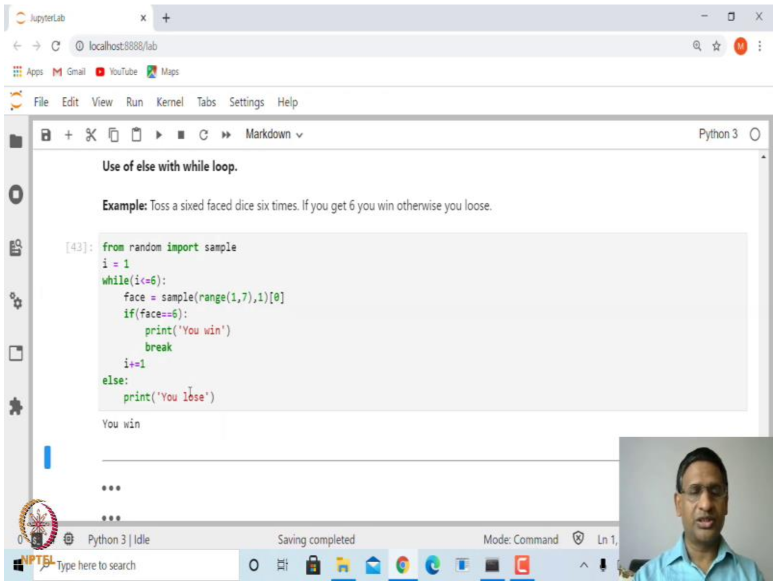
Example: Toss a sixed faced dice six times. If you get 6 you win otherwise you loose.

```
[42]: from random import sample
i = 1
while(i<=6):
    face = sample(range(1,7),1)[0]
    if(face==6):
        print('You win')
        break
    i+=1
else:
    print('You lose')
```

```
You win
```

If I run again, you win.

(Refer Slide Time: 27:26)



Use of else with while loop.

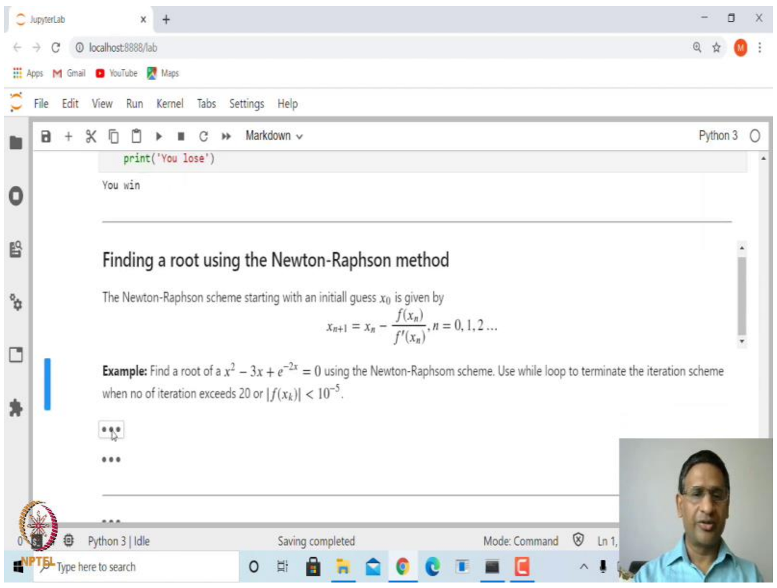
Example: Toss a sixed faced dice six times. If you get 6 you win otherwise you loose.

```
[43]: from random import sample
i = 1
while(i<=6):
    face = sample(range(1,7),1)[0]
    if(face==6):
        print('You win')
        break
    i+=1
else:
    print('You lose')
```

```
You win
```

Again, you win. So, what it says, is most likely when you toss 6 times one of the thing will you will get as 6.

(Refer Slide Time: 27:34)



```python
    print('You lose')
```

```
You win
```

Finding a root using the Newton-Raphson method

The Newton-Raphson scheme starting with an initiall guess x_0 is given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n = 0, 1, 2 \ldots$$

Example: Find a root of a $x^2 - 3x + e^{-2x} = 0$ using the Newton-Raphsom scheme. Use while loop to terminate the iteration scheme when no of iteration exceeds 20 or $|f(x_k)| < 10^{-5}$.

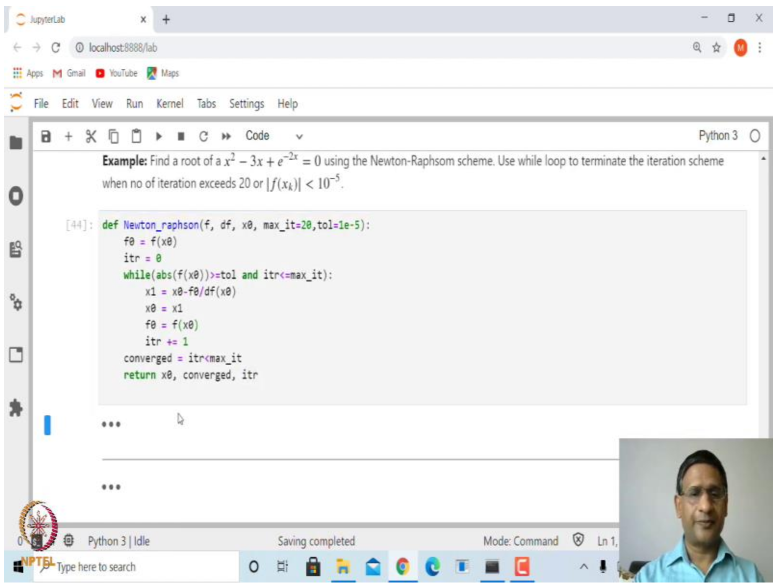
Let us look at one last example, of finding root using Newton-Raphson method. I am sure this you must have already seen. So, if you have a function f(x) equal to 0 and you want to find a root of this, that means, find a value of x such that f(x) equals to 0, one can use what is known as Newton-Raphson method.

So, you start with some initial guess value x0. Of course, you need to know the convergence criteria for this, not every initial guess, and not every function will have the convergence. But at present I am not telling, you what should be the convergence criteria and all that. I hope you already know it. So, this is the Newton-Raphson scheme.

You start with x0, and then define x n plus 1. For n equal to 0, x n plus 1 is x1, which is x0 minus f at f at x0 divided by f dash at x0, of course you need the function to be differentiable. And the next time you once you get x1 you find out x2 and so on.

So, let us make use of this iteration scheme for f(x) equal to x square minus 3x plus e to the power minus 2 x. And in this case, let us make use of while loop, and we do not want to do too many iterations. So, maximum 20 iterations or if the value of the function becomes less than 10 to the power minus 5. This is an absolute value because the function value can be negative. So, in absolute value if it is less than 10 to the power minus 5 you stop. So, how do we do this?

(Refer Slide Time: 29:13)



Example: Find a root of a $x^2 - 3x + e^{-2x} = 0$ using the Newton-Raphsom scheme. Use while loop to terminate the iteration scheme when no of iteration exceeds 20 or $|f(x_k)| < 10^{-5}$.

```
[44]: def Newton_raphson(f, df, x0, max_it=20,tol=1e-5):
          f0 = f(x0)
          itr = 0
          while(abs(f(x0))>=tol and itr<=max_it):
              x1 = x0-f0/df(x0)
              x0 = x1
              f0 = f(x0)
              itr += 1
          converged = itr<max_it
          return x0, converged, itr
```

So, let us write a user defined function for this. So, the name I am giving is Newton-Raphson and you input f, you need df, that is, derivative of f, initial guess, maximum number of iteration, as I said is 20 and the tolerance which is the 10 to the power minus 5 in this case.

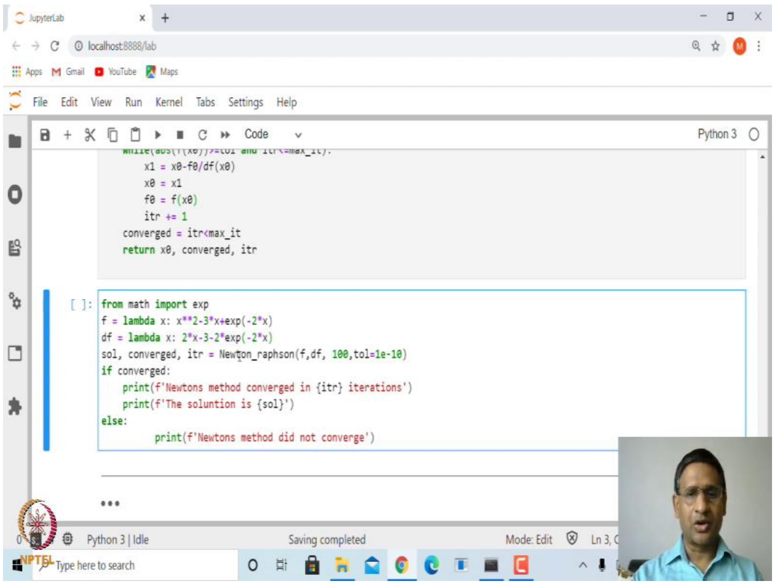
So, once you have entered this, define f0 to be f at x0, iteration number is 0 and apply while loop. So, while the absolute value of f(x) is bigger than the tolerance, because we want in case the tolerance is f(xk) is less than 10 to the power minus 5 k then you stop. Therefore this has to run as long as the value of the function in absolute value is bigger than tolerance. And the number of iteration is less than equal to 20.

So, if if both these conditions are satisfied then it has to run. If any one of this is false, that means, if this value becomes less than the tolerance or this value becomes the iteration becomes more than 20 then you stop. And then what you do? You define x1 to be x0 minus f at x0 divided by df at x0, and then, again you value of x1 you put it in x0, f at x0, you recalculate and increase the iteration number.

And at the end I am just saying that in case the number of iteration is less than the max_it then it has converged. If that does not happen then it has not converged. So, this is going to be true in case the convergence is obtained within 20 iterates and false otherwise.

So, and it returns the value x0 that is, the x n plus 1, and then true or false depending upon whether it converges or not and then in case it converges the number of iterations,   So, let us run this.

(Refer Slide Time: 31:12)



```
        x1 = x0-f0/df(x0)
        x0 = x1
        f0 = f(x0)
        itr += 1
    converged = itr<max_it
    return x0, converged, itr
```

```
[ ]: from math import exp
     f = lambda x: x**2-3*x+exp(-2*x)
     df = lambda x: 2*x-3-2*exp(-2*x)
     sol, converged, itr = Newton_raphson(f,df, 100,tol=1e-10)
     if converged:
         print(f'Newtons method converged in {itr} iterations')
         print(f'The solunstion is {sol}')
     else:
             print(f'Newtons method did not converge')
```

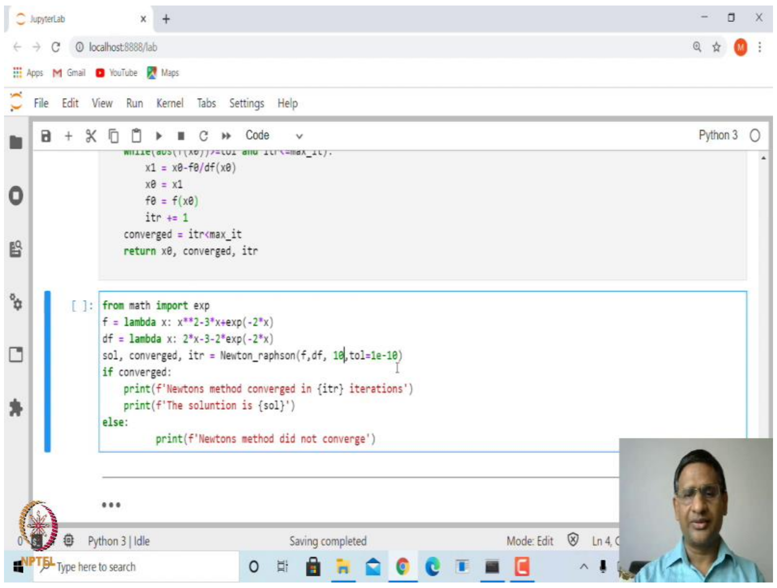
Next let us call this function. So, let us first define this function. We are defining this function using not def, but using lambda.

So, this is another convenient way of defining a function quickly. So, this is f is equal to lambda, f is the name of the function and this is the argument, and the output that is returned which is x to the power 2 minus 2 x plus exponential of   minus 2 x and that is why we are importing exponential.

Similarly you define derivative of the function in df, again using lambda function.

And then call this function Newton-Raphson, f, df, and initial guess, let us say, I make initial guess is equal to 10, and the tolerance limit I have is equal to 5.

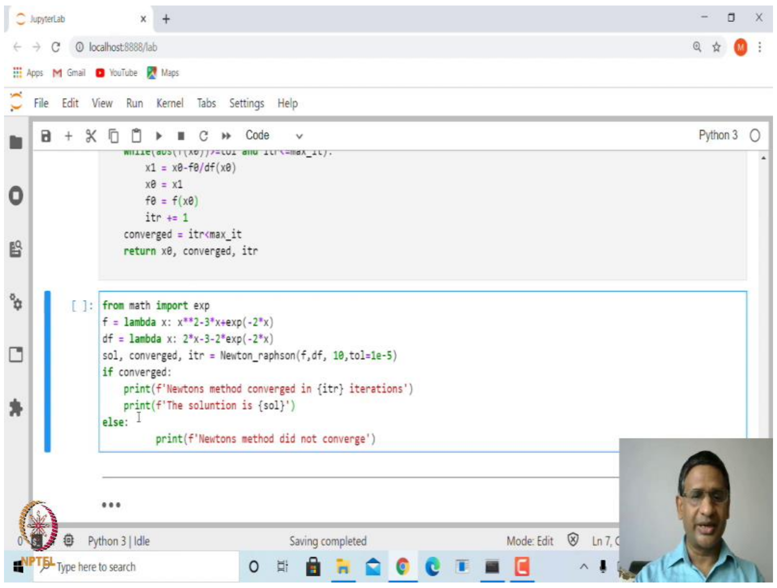
(Refer Slide Time: 31:59)



```
    while(abs(f(x0))>=tol and itr<=max_it):
        x1 = x0-f0/df(x0)
        x0 = x1
        f0 = f(x0)
        itr += 1
    converged = itr<max_it
    return x0, converged, itr
```

```
[ ]: from math import exp
     f = lambda x: x**2-3*x+exp(-2*x)
     df = lambda x: 2*x-3-2*exp(-2*x)
     sol, converged, itr = Newton_raphson(f,df, 10,tol=1e-10)
     if converged:
         print(f'Newtons method converged in {itr} iterations')
         print(f'The solution is {sol}')
     else:
             print(f'Newtons method did not converge')
```

(Refer Slide Time: 32:04)

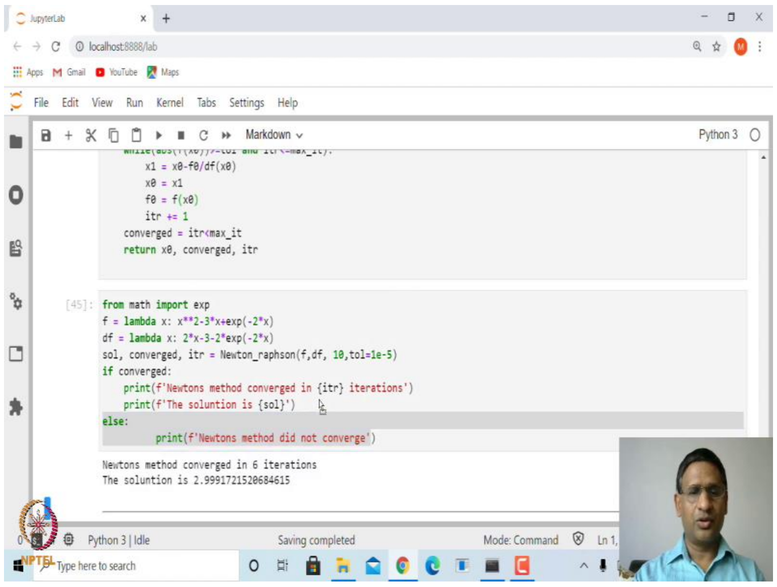


```
    while(abs(f(x0))>=tol and itr<=max_it):
        x1 = x0-f0/df(x0)
        x0 = x1
        f0 = f(x0)
        itr += 1
    converged = itr<max_it
    return x0, converged, itr
```

```
[ ]: from math import exp
     f = lambda x: x**2-3*x+exp(-2*x)
     df = lambda x: 2*x-3-2*exp(-2*x)
     sol, converged, itr = Newton_raphson(f,df, 10,tol=1e-5)
     if converged:
         print(f'Newtons method converged in {itr} iterations')
         print(f'The solution is {sol}')
     else:
             print(f'Newtons method did not converge')
```

And then let us print in case it converges, if converged; that means, if it is true value then you print that the Newton methods have converged and with so many iterations. And also print what is the solution you have got otherwise, you print that it has not converged.

(Refer Slide Time: 32:27)



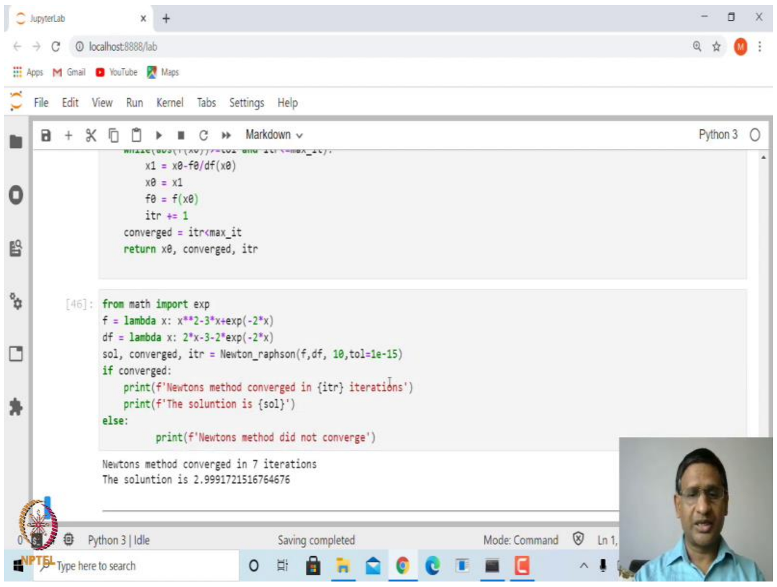
```
        x1 = x0-f0/df(x0)
        x0 = x1
        f0 = f(x0)
        itr += 1
    converged = itr<max_it
    return x0, converged, itr
```

```
[45]: from math import exp
f = lambda x: x**2-3*x+exp(-2*x)
df = lambda x: 2*x-3-2*exp(-2*x)
sol, converged, itr = Newton_raphson(f,df, 10,tol=1e-5)
if converged:
    print(f'Newtons method converged in {itr} iterations')
    print(f'The solution is {sol}')
else:
        print(f'Newtons method did not converge')
```

```
Newtons method converged in 6 iterations
The solution is 2.9991721520684615
```

So, let us run this. When you run this it says that it this has converged in 6 iteration. If I increase this, let us say, if I make it 15, then it has convergence in 6.

(Refer Slide Time: 32:34)



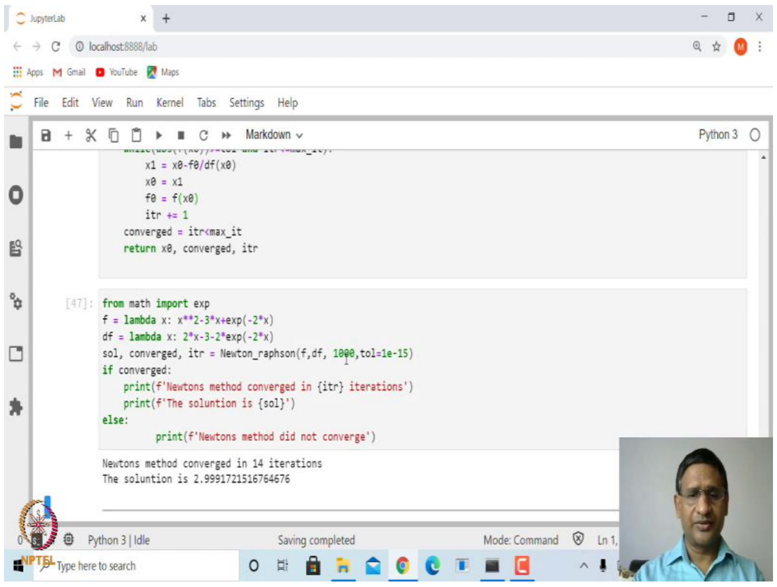
```
        x1 = x0-f0/df(x0)
        x0 = x1
        f0 = f(x0)
        itr += 1
    converged = itr<max_it
    return x0, converged, itr
```

```
[46]: from math import exp
f = lambda x: x**2-3*x+exp(-2*x)
df = lambda x: 2*x-3-2*exp(-2*x)
sol, converged, itr = Newton_raphson(f,df, 10,tol=1e-15)
if converged:
    print(f'Newtons method converged in {itr} iterations')
    print(f'The solution is {sol}')
else:
        print(f'Newtons method did not converge')
```

```
Newtons method converged in 7 iterations
The solution is 2.9991721516764676
```

And if I make it let us say the initial guess very far, let us say 1000 then it has required 14 iterations.

(Refer Slide Time: 32:41)

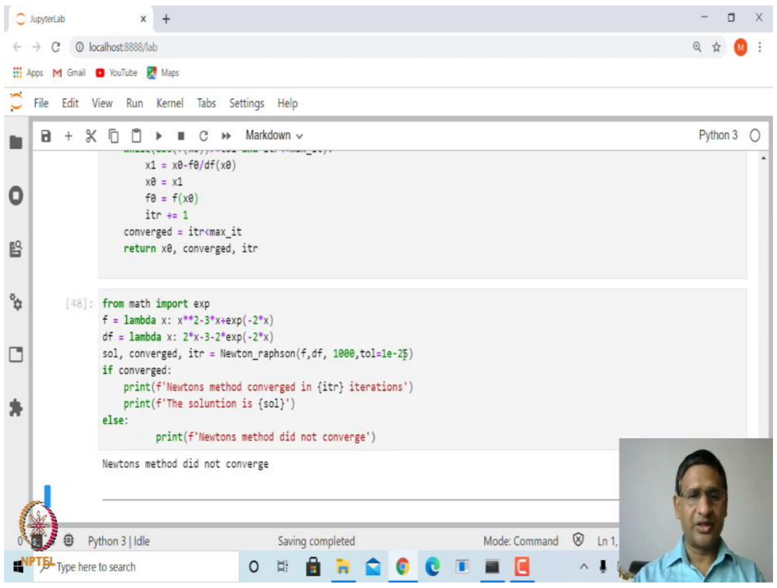


```
        x1 = x0-f0/df(x0)
        x0 = x1
        f0 = f(x0)
        itr += 1
    converged = itr<max_it
    return x0, converged, itr
```

```
[47]: from math import exp
f = lambda x: x**2-3*x+exp(-2*x)
df = lambda x: 2*x-3-2*exp(-2*x)
sol, converged, itr = Newton_raphson(f,df, 1000,tol=1e-15)
if converged:
    print(f'Newtons method converged in {itr} iterations')
    print(f'The soluntion is {sol}')
else:
        print(f'Newtons method did not converge')
```

```
Newtons method converged in 14 iterations
The soluntion is 2.9991721516764676
```

(Refer Slide Time: 32:50)



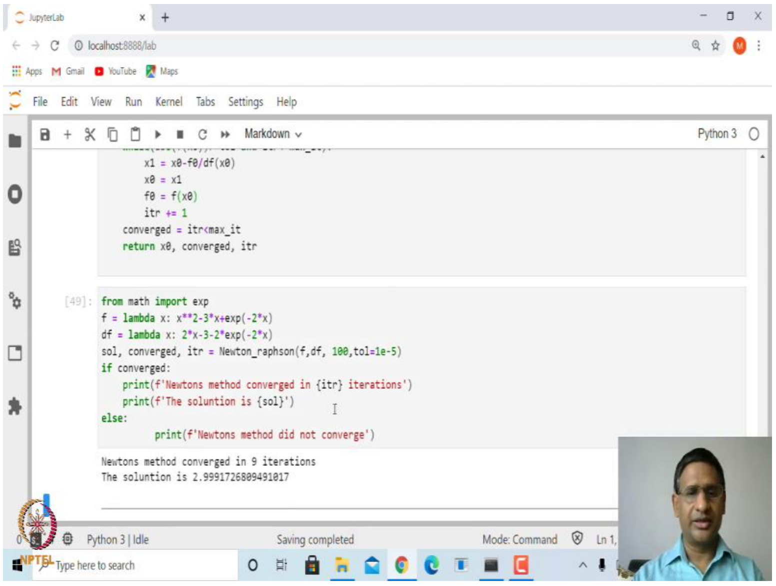
```
        x1 = x0-f0/df(x0)
        x0 = x1
        f0 = f(x0)
        itr += 1
    converged = itr<max_it
    return x0, converged, itr
```

```
[48]: from math import exp
f = lambda x: x**2-3*x+exp(-2*x)
df = lambda x: 2*x-3-2*exp(-2*x)
sol, converged, itr = Newton_raphson(f,df, 1000,tol=1e-25)
if converged:
    print(f'Newtons method converged in {itr} iterations')
    print(f'The soluntion is {sol}')
else:
        print(f'Newtons method did not converge')
```

```
Newtons method did not converge
```

If instead, let us say if I make it 25 then it may, says that it has not converged. That means, in case, this is very very small tolerance limit it will require more than 20 iterates.

(Refer Slide Time: 33:02)



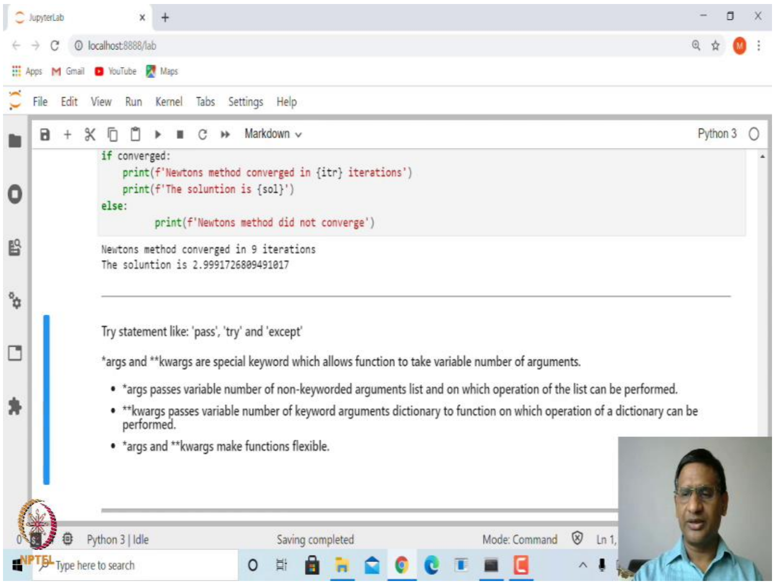
```
        x1 = x0-f0/df(x0)
        x0 = x1
        f0 = f(x0)
        itr += 1
    converged = itr<max_it
    return x0, converged, itr
```

```
[49]: from math import exp
f = lambda x: x**2-3*x+exp(-2*x)
df = lambda x: 2*x-3-2*exp(-2*x)
sol, converged, itr = Newton_raphson(f,df, 100,tol=1e-5)
if converged:
    print(f'Newtons method converged in {itr} iterations')
    print(f'The soluntion is {sol}')
else:
        print(f'Newtons method did not converge')
```

```
Newtons method converged in 9 iterations
The soluntion is 2.9991726809491017
```

So, let me   write here 5 and maybe here 100, so it has converged. So, these are the examples using while loop.

(Refer Slide Time: 33:14)



```
if converged:
    print(f'Newtons method converged in {itr} iterations')
    print(f'The soluntion is {sol}')
else:
        print(f'Newtons method did not converge')
```

```
Newtons method converged in 9 iterations
The soluntion is 2.9991726809491017
```

Try statement like: 'pass', 'try' and 'except'

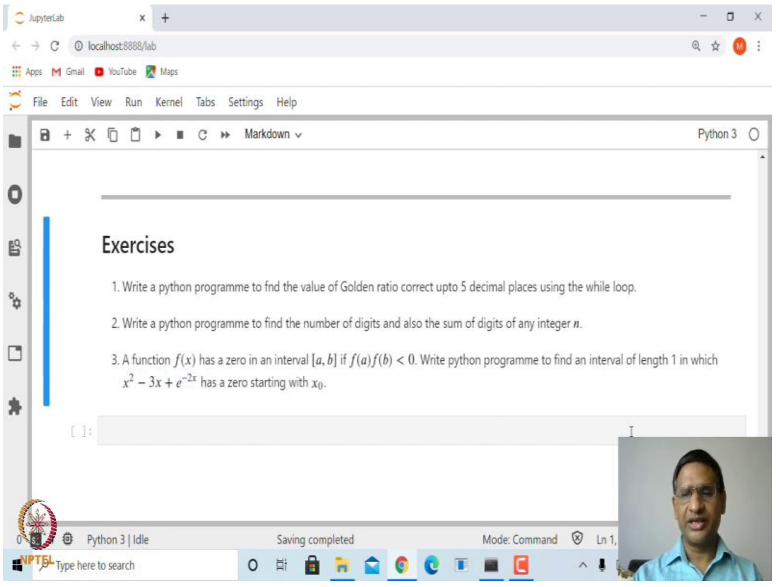
*args and **kwargs are special keyword which allows function to take variable number of arguments.

- *args passes variable number of non-keyworded arguments list and on which operation of the list can be performed.
- **kwargs passes variable number of keyword arguments dictionary to function on which operation of a dictionary can be performed.
- *args and **kwargs make functions flexible.

Now, you should also look at, in Python you also have a notion of pass along with loops, you can also look at how to make use of try and accept. When we have created this user defined function we mostly give the fixed number of parameters. Of course, some of them are default parameter for example, in this case max_it and tolerance. But in case you do not know how many parameters user is going to supply then you can make use of what is known as *args and **kwargs. These two are very convenient way to make function flexible.

*args passes a variable number of non-keyworded arguments. So, it will be non-keyworded argument. And then you can apply operations on all these keywords. And similarly, **kwargs, here args and kwargs you can replace it by some other thing also. And in this case this is actually a keyworded argument something very similar to dictionary. So, try to explore these options, they are very useful and it makes functions flexible.

(Refer Slide Time: 34:38)



Exercises

1. Write a python programme to fnd the value of Golden ratio correct upto 5 decimal places using the while loop.
2. Write a python programme to find the number of digits and also the sum of digits of any integer n.
3. A function $f(x)$ has a zero in an interval $[a, b]$ if $f(a)f(b) < 0$. Write python programme to find an interval of length 1 in which $x^2 - 3x + e^{-2x}$ has a zero starting with x_0.

At the end let me leave you with few exercises. These are simple exercises.

The first exercise is write a python programme to find the value of Golden ratio correct up to 5 decimal places using while loop. I have already defined what is golden ratio in the previous lecture.

The second exercise is to write a python programme to find the number of digits and also the sum of digits of any integer n. So, n could be any integer, could be small or very large.

The third one is, a function f(x) has a 0, in an interval a, b, if the product of f(a) and f(b) is less than 0, that means, the value of the function at a and b are of opposite signs. And so, what I want you to do. Look at this function f(x), it is the same function for which we wrote Newton-Raphson method and we know that it has a 0.

So, find an interval of length 1 which contains 0 of this function, that is the third exercise. So, these are the 3 exercises. Of course, we will be posting the solution of these exercises after some time. But you should try, these are all very simple exercises,

So, let me finish here. So, thank you very much. We will see you in the next lecture.