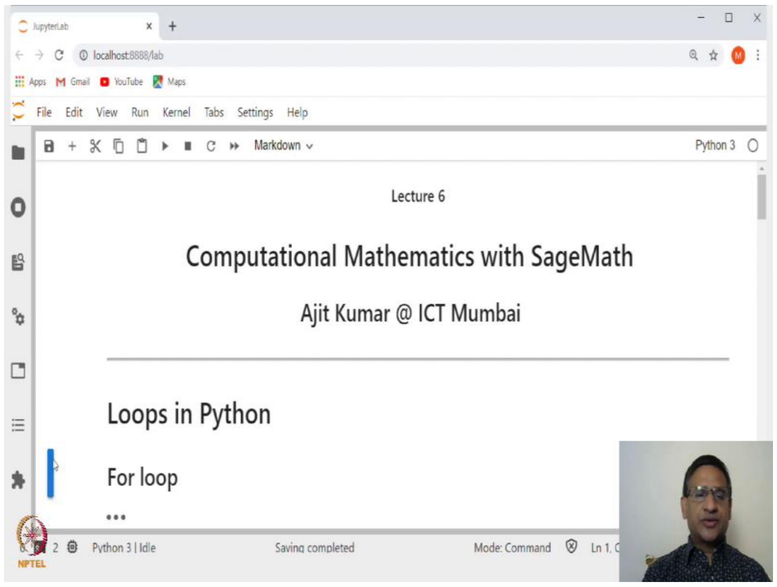
Computational Mathematics with Sagemath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Lecture – 07
For loop in Python

Welcome to the 6th lecture on Computational Mathematics with Sagemath. In the next 2 lectures we will be looking at loops in Python. Whenever you want to do some task repeatedly. You want to execute some statement, whenever certain list of conditions are satisfied, we use loops.

(Refer Slide Time: 00:47)


Lecture 6

Computational Mathematics with SageMath

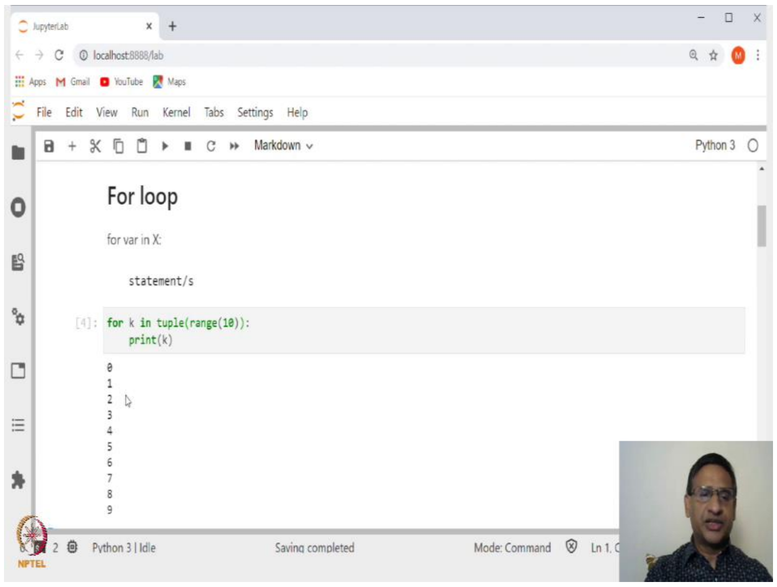
Ajit Kumar @ ICT Mumbai

Loops in Python

For loop

Welcome to the 6th lecture on Computational Mathematics with Sagemath. In the next 2 lectures we will be looking at loops in Python. Whenever you want to do some task repeatedly. You want to execute some statement, whenever certain list of conditions are satisfied, we use loops.

(Refer Slide Time: 00:51)



For loop

for var in X:

    statement/s

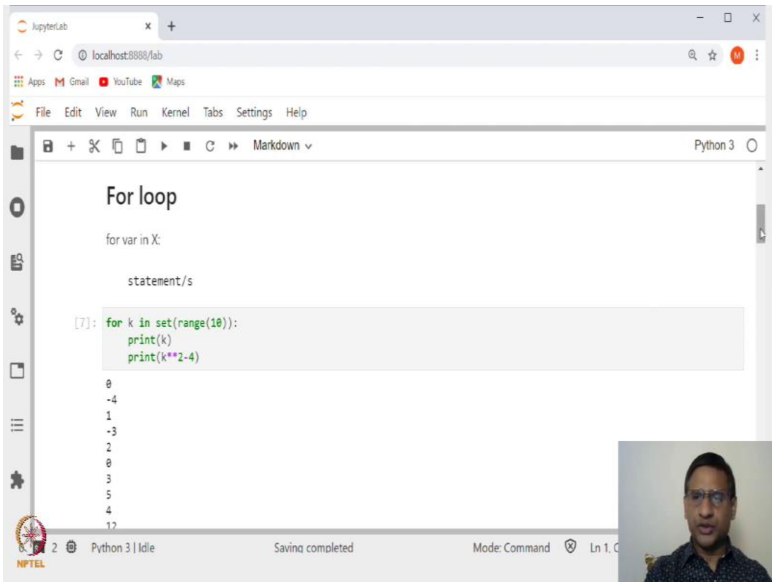
```
[4]: for k in tuple(range(10)):
         print(k)
```

```
0
1
2
3
4
5
6
7
8
9
```

We will look at few examples. For example, let us say, we have, for let us say variable k in let us say range 10. So, range 10 means, it will give you 0 to 10. Suppose, I just want to print what is k. So, what it is going to do it? It will take value k in this set range 10, but range 10 is 0 to 9, it is a list. So, for each k it will print that element. So, first it will print 0, then it will print 1, then 2 and so on in in one line.

Now, this as I said, instead of range, if I make it let us say tuple, that means, we are creating a tuple of number 0 to 10. If I run this again, you will see that it gives you the same thing. So, this X can be list and it can also be tuple.

(Refer Slide Time: 02:38)



For loop

for var in X:

```
    statement/s
```

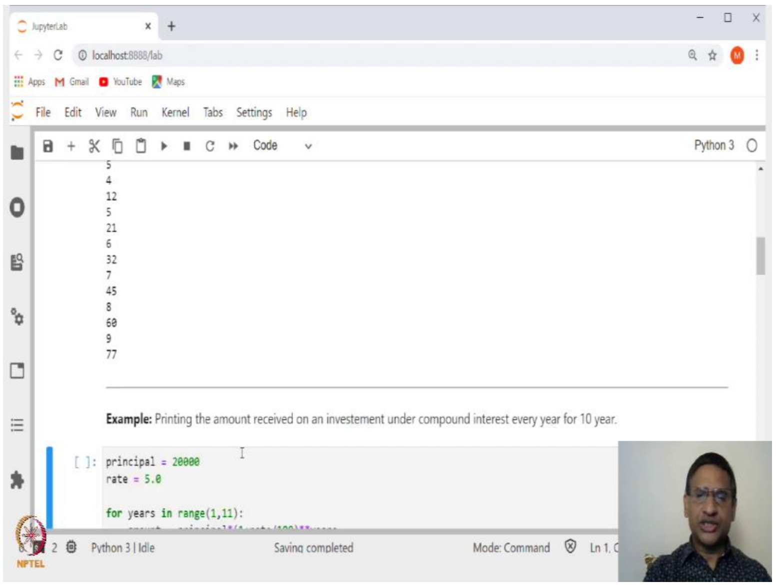
```
[7]: for k in set(range(10)):
         print(k)
         print(k**2-4)
```

```
0
-4
1
-3
2
0
3
5
4
12
```

How about suppose we make this as a set? If, I make this as a set, then still it works same way. We will later on we will see it can also be a dictionary. Basically we will be having this X as list mostly, and we have to iterate over that list.

Here we had only one statement, but you can have multiple statements which is executed. For example, let us say print(k) and also let us say print k square minus 4, sorry k square should be double star minus 4. So, you can see here for each k it is first printing 0, 0 square minus 4 is minus 4 and so on.

(Refer Slide Time: 03:32)



```
5
4
12
5
21
6
32
7
45
8
60
9
77
```

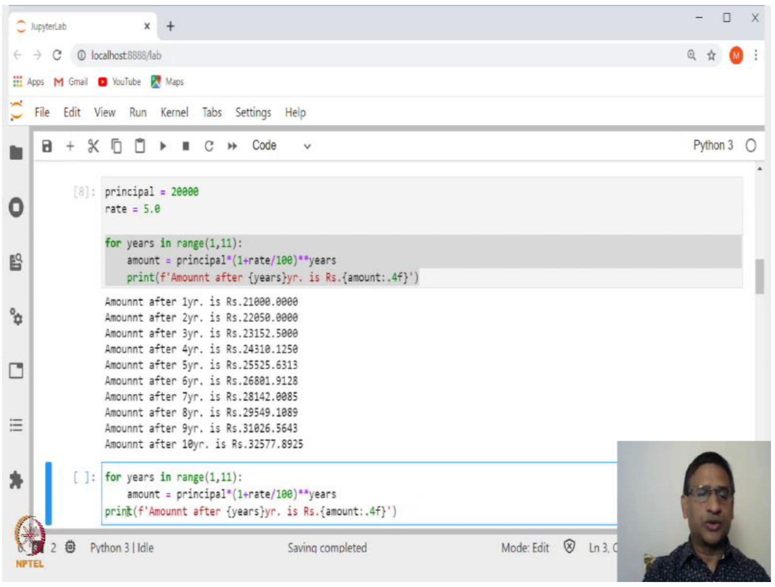
Example: Printing the amount received on an investement under compound interest every year for 10 year.

```
[ ]: principal = 20000
     rate = 5.0

     for years in range(1,11):
```

This is a very simple for loop. Now, let us look at some examples, how to make use of this for loop through examples.

For the first example, we have already seen, how to compute the amount received, when we it is invested under compound interest, let us say on some interest rate. Suppose we want to print the amount received on an investment, under compound interest every year for first 10 years.

(Refer Slide Time: 04:10)



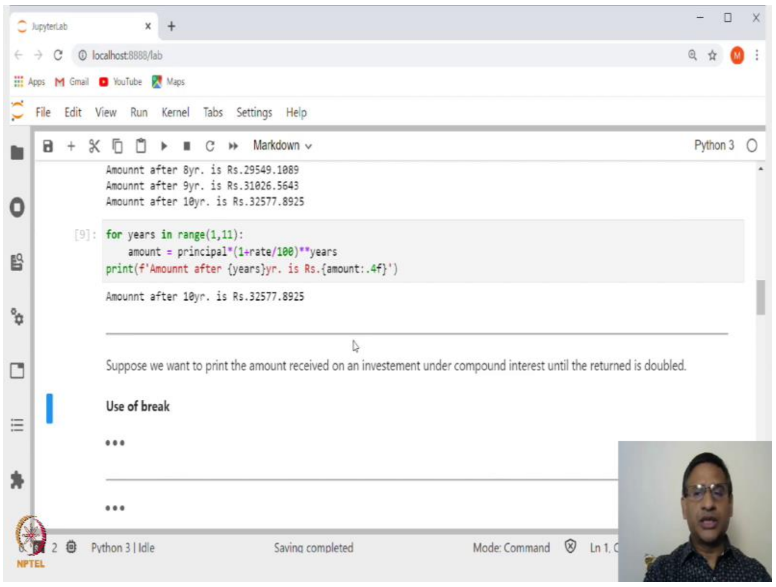
```
[8]: principal = 20000
     rate = 5.0

     for years in range(1,11):
         amount = principal*(1+rate/100)**years
         print(f'Amounnt after {years}yr. is Rs.{amount:.4f}')

     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
     Amounnt after 9yr. is Rs.31026.5643
     Amounnt after 10yr. is Rs.32577.8925

[ ]: for years in range(1,11):
         amount = principal*(1+rate/100)**years
     print(f'Amounnt after {years}yr. is Rs.{amount:.4f}')
```

Let us see, how do we do that? Just recall that how we did this earlier? If the principal is, let us say, 20,000 and interest rate is 5 percent, and then we want to print the amount received after every 1 year. So, after 1 year so that means, that year will vary from 1 to 10. So, 1 to 10, if I have to generate, the range function, we have to say range 1 comma, 11 and then colon and then calculate the amount. This amount is principal into 1 plus rate divided by 100 to the power number of years and years will take value 1 to 10.

And, then you just print the amount after so many years is given by this using f string. If I execute this you will see that, it will give you the amount received after 1 year as 21000, after 2 years 22050 and so on. We have printed 4 decimal places that is why here you have dot 0.4f.

So, this is how you can make use of for loop in order to print all the amount received after every year. Suppose, we wanted to print only after 10 years, of course, for that we may not even need for loop, but in that case this print statement, which you have here you can push it outside this loop. So, let me just copy paste this and let us insert 1 input cell after this. So, if I push this outside this for loop, then what is it doing it is computing the amount after every year. And, after end of this loop once this loop is over, then amount is calculated after 10 years. And, the only the last one will be printed. What you see here only this last one will be printed. This you can see here. That is correct. After 10 years the amount received is this much right.

(Refer Slide Time: 06:21)



```
Amounnt after 8yr. is Rs.29549.1089
Amounnt after 9yr. is Rs.31026.5643
Amounnt after 10yr. is Rs.32577.8925
```

```
[9]: for years in range(1,11):
         amount = principal*(1+rate/100)**years
     print(f'Amounnt after {years}yr. is Rs.{amount:.4f}')
```

```
Amounnt after 10yr. is Rs.32577.8925
```

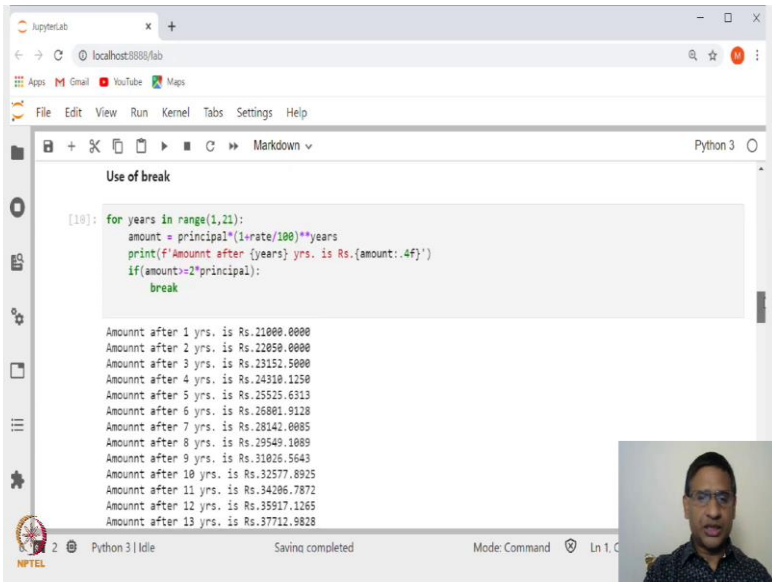
Suppose we want to print the amount received on an investement under compound interest until the returned is doubled.

Use of break

Now, suppose it so happen that in between you want to break the loop. So, for example, here we have printed for every years for 10 years. Instead of 10 years, if you want 20 years, you can put here 21. But, suppose I want to take out the money, after it has become doubled. I do not want to be too greedy, I want to invest only till the money becomes double.

In that case we do not know how many years it will take? How many years it will take? Of course, one can calculate number of years based on this formula. One can obtained a formula for that. But let us say we do not know how many years it is going to take, when we invest under compound interest. So, what we will do? We will be using what is called break statement inside for loop. If you want to break a loop in between, whenever some condition is satisfied one can use break.

(Refer Slide Time: 07:42)



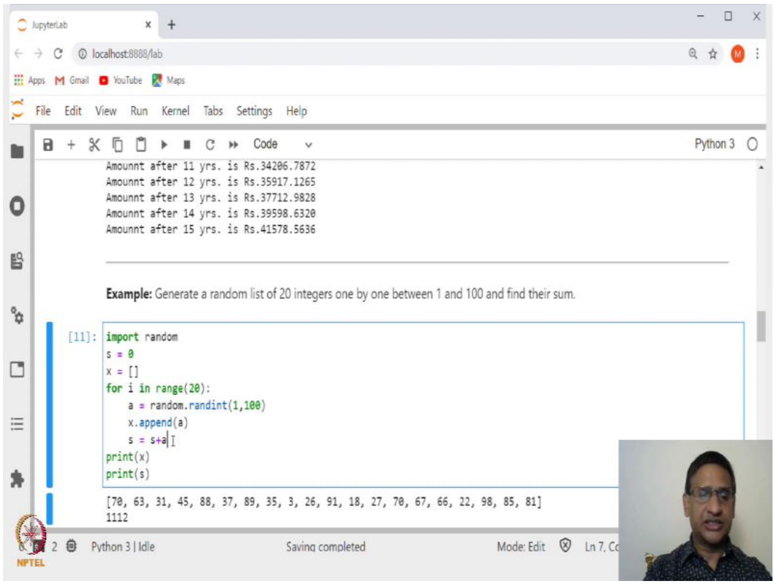
Use of break

```
[10]: for years in range(1,21):
          amount = principal*(1+rate/100)**years
          print(f'Amounnt after {years} yrs. is Rs.{amount:.4f}')
          if(amount>=2*principal):
              break
```

```
Amounnt after 1 yrs. is Rs.21000.0000
Amounnt after 2 yrs. is Rs.22050.0000
Amounnt after 3 yrs. is Rs.23152.5000
Amounnt after 4 yrs. is Rs.24310.1250
Amounnt after 5 yrs. is Rs.25525.6313
Amounnt after 6 yrs. is Rs.26801.9128
Amounnt after 7 yrs. is Rs.28142.0085
Amounnt after 8 yrs. is Rs.29549.1089
Amounnt after 9 yrs. is Rs.31026.5643
Amounnt after 10 yrs. is Rs.32577.8925
Amounnt after 11 yrs. is Rs.34206.7872
Amounnt after 12 yrs. is Rs.35917.1265
Amounnt after 13 yrs. is Rs.37712.9828
```

How do we do that? What you need to do is, again the same command. So, let us say we do not know, how many years it is going to take. So, we will run for loop, let us say for 20 years. So, we make it 21 and the amount received, then print that amount. And, in case the amount exceeds or if it becomes greater than equal to 2 times the principal amount then you break. So, again here you have colon and then followed by break.

(Refer Slide Time: 08:19)



```
Amounnt after 11 yrs. is Rs.34206.7872
Amounnt after 12 yrs. is Rs.35917.1265
Amounnt after 13 yrs. is Rs.37712.9828
Amounnt after 14 yrs. is Rs.39598.6320
Amounnt after 15 yrs. is Rs.41578.5636
```

Example: Generate a random list of 20 integers one by one between 1 and 100 and find their sum.

```
[11]: import random
      s = 0
      x = []
      for i in range(20):
          a = random.randint(1,100)
          x.append(a)
          s = s+a
      print(x)
      print(s)
```

```
[70, 63, 31, 45, 88, 37, 89, 35, 3, 26, 91, 18, 27, 70, 67, 66, 22, 98, 85, 81]
1112
```

So, let us execute this, if I execute this, it is printing 1st year, 2nd year, 3rd year, 4th year and so on. And, and after 15 years if the amount has become 41578, which is the more than double of 20000 and in the previous year, 14th years the amount was just less than the double.

So, this is how we can make use of break statement, in order to break the loop at intermediate stage, whenever some condition is satisfied. This is very useful.

Let us look at some other example, suppose we want to generate a random list of 20 integers, let us say one at a time put it in some list and these random numbers are from 1 to 100 and then we want to find their sum.

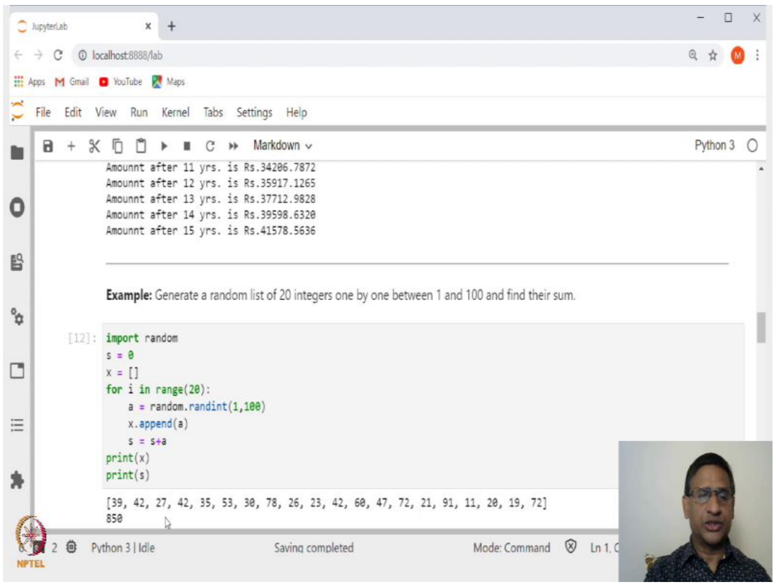
So, it is like, you generate one number, let us say the 1st number is 5, next number is let us say 7. So, you add 5 plus 7 that is 12. Generate next number, say 51, then 51 plus12 becomes 63 and so on. So, keep on doing this how many times? 20 times.

So, let us see how we can make use of for loop for this. So, first we will use the module random. So import random, and inside that random there is a function called random, we have seen earlier. And that random integer, we want between 1 and 100.

So, first we generate, let us say empty list and in that empty list, we will keep on appending the number which we have generated. So, a is equal to random dot randint 1 comma 100. So, it will generate one random integer between 1 and 100 and then you append this in a, and then at the same time you add this to s, s initial value is 0.

So, so for example, s is 0 and it will generate first number, as I said first number could be let us say 5, then this list becomes singleton 5 and then s becomes s plus 5, s is 0. Therefore, a is 5. So, s becomes now 5. And, it will continue till this I exhausts all the elements in this list that is 20 times it will run, and at the end you print the list. So, the list is 70, 63, 31 and so on and the sum of these number is 1112.

(Refer Slide Time: 11:00)



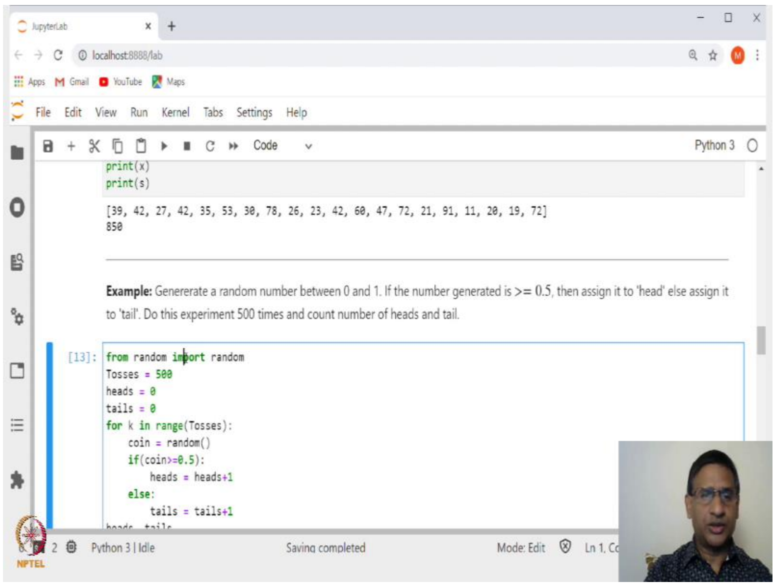
```
Amounnt after 11 yrs. is Rs.34206.7872
Amounnt after 12 yrs. is Rs.35917.1265
Amounnt after 13 yrs. is Rs.37712.9828
Amounnt after 14 yrs. is Rs.39598.6320
Amounnt after 15 yrs. is Rs.41578.5636
```

Example: Generate a random list of 20 integers one by one between 1 and 100 and find their sum.

```
[12]: import random
s = 0
x = []
for i in range(20):
    a = random.randint(1,100)
    x.append(a)
    s = s+a
print(x)
print(s)

[39, 42, 27, 42, 35, 53, 30, 78, 26, 23, 42, 60, 47, 72, 21, 91, 11, 20, 19, 72]
850
```

Suppose, if I run this once again, I will get a different set of random integers as a list and the sum will also be different, that is what you can see. So, this is how we are able to generate 20 random numbers. Instead of 20 random numbers you can have 2000 random numbers and still run this.

(Refer Slide Time: 11:18)



```
print(x)
print(s)

[39, 42, 27, 42, 35, 53, 30, 78, 26, 23, 42, 60, 47, 72, 21, 91, 11, 20, 19, 72]
850
```

Example: Genererate a random number between 0 and 1. If the number generated is >= 0.5, then assign it to 'head' else assign it to 'tail'. Do this experiment 500 times and count number of heads and tail.

```
[13]: from random import random
Tosses = 500
heads = 0
tails = 0
for k in range(Tosses):
    coin = random()
    if(coin>=0.5):
        heads = heads+1
    else:
        tails = tails+1
```

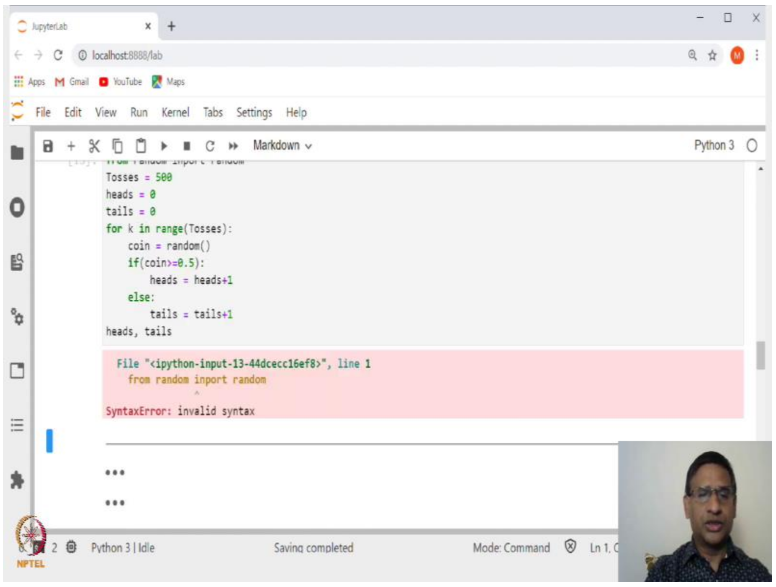
Next, let us look at another example. Suppose we want to generate a random number between 0 and 1. So, it will be decimal number between 0 and 1. And, suppose if the random number generated is more than 0.5, that is greater than equal to 0.5, then we will call that as let us say head otherwise, we will call as tail.

Now, suppose we do this experiment 500 times and count number of times you have obtained heads and tails. This is just a an experiment, which you are doing. Instead of tossing a coin, you are generating a random number between a 0 and 1. When the number is greater than equal to 0.5, you assign that as a head otherwise tail. So, this is very simple way to look at.

So, number of tosses is 500, that is 500. And, let us say to begin with number of head is 0 and number of tails is also 0. Now, run a loop k in this range tosses; that means, it will generate 0 to 500. And, then let us call the number as a coin and this random number between 0 and 1 can be generated by using the function random, from random library, we have already imported random library.

But, it may be a good idea to import it again, because sometimes you may be running this it after opening this file, and you may not have be imported. So, let me just call from random import random. So, now I do not need to say here dot random.

(Refer Slide Time: 13:11)

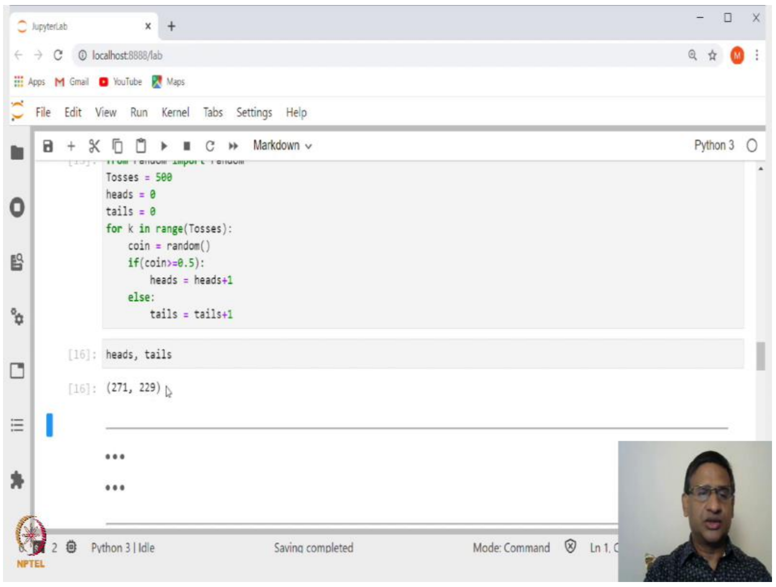


```
Tosses = 500
heads = 0
tails = 0
for k in range(Tosses):
    coin = random()
    if(coin>=0.5):
        heads = heads+1
    else:
        tails = tails+1
heads, tails
```

```
  File "<ipython-input-13-44dcecc16ef8>", line 1
    from random inport random
                     ^
SyntaxError: invalid syntax
```

I think there is spelling mistake from random import. What we are doing?

(Refer Slide Time: 13:20)



```
Tosses = 500
heads = 0
tails = 0
for k in range(Tosses):
    coin = random()
    if(coin>=0.5):
        heads = heads+1
    else:
        tails = tails+1
```

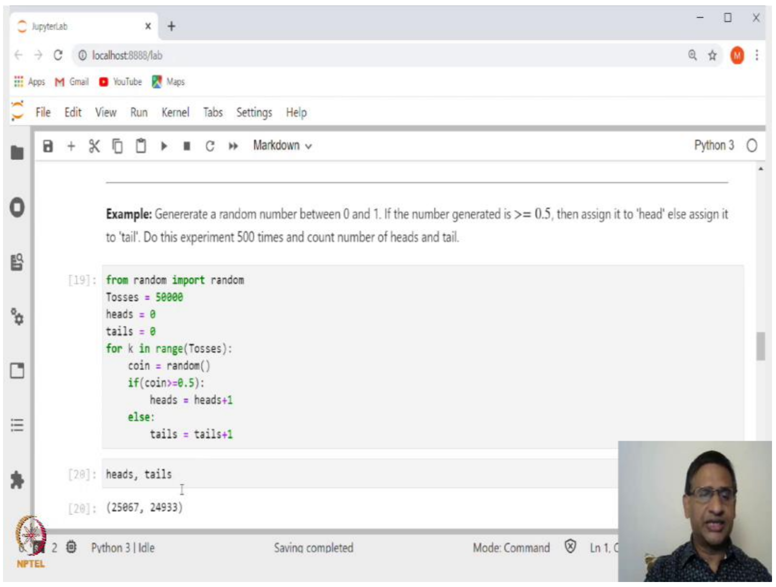
```
[16]: heads, tails
```

```
[16]: (271, 229)
```

So, call this number as coin. And, in case this number is greater than equal to 0.5 then increase head by 1, otherwise increase tail by 1 and then you can print after this loop. So, heads and tails. So, heads are 271 tails is299.

 Suppose instead of 500 if I do this experiment 5000 times. And, look at how many heads and tails. this is 2481 this is 2519. So, almost 50 percent.

(Refer Slide Time: 14:14)



Example: Genererate a random number between 0 and 1. If the number generated is >= 0.5, then assign it to 'head' else assign it to 'tail'. Do this experiment 500 times and count number of heads and tail.

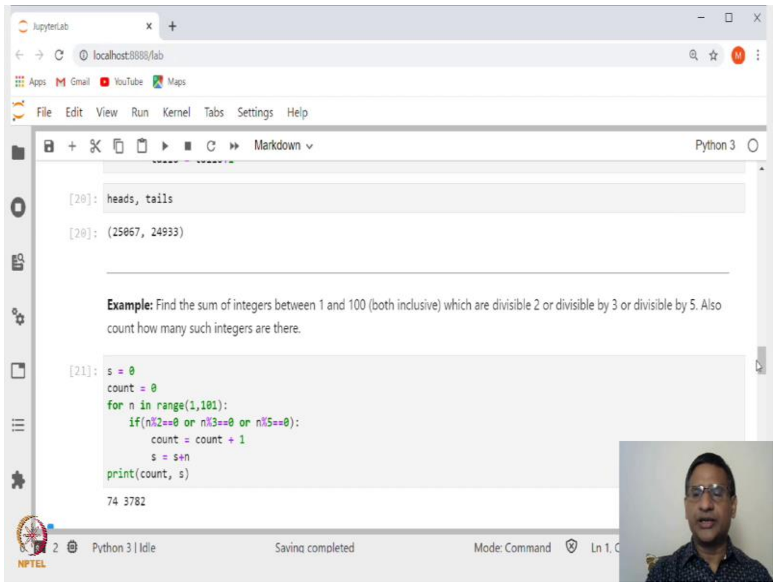
```
[19]: from random import random
Tosses = 50000
heads = 0
tails = 0
for k in range(Tosses):
    coin = random()
    if(coin>=0.5):
        heads = heads+1
    else:
        tails = tails+1
```

```
[20]: heads, tails
```

```
[20]: (25067, 24933)
```

If I increased to even bigger number of tosses. And, then see how many heads and tails? So, it will be almost close to 50 percent. That is why the probability of heads when you toss a coin is 0.5. So, this is an experiment.

(Refer Slide Time: 14:29)



```
[20]: heads, tails

[20]: (25067, 24933)
```

Example: Find the sum of integers between 1 and 100 (both inclusive) which are divisible 2 or divisible by 3 or divisible by 5. Also count how many such integers are there.

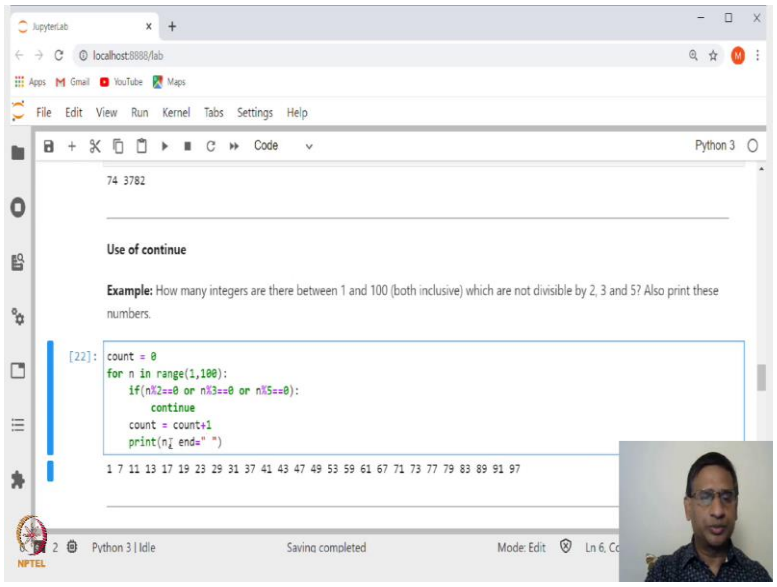
```
[21]: s = 0
      count = 0
      for n in range(1,101):
          if(n%2==0 or n%3==0 or n%5==0):
              count = count + 1
              s = s+n
      print(count, s)

      74 3782
```

Let us look at another example, suppose we want to find the sum of integers between 1 and 100 both included, which are divisible by 2 or 3 or 5. So, all those integers, which are divisible by 2, or 3, or 5. Then add all these between 1 and 100 and also count how many such integers are there?

So, in this case we can do using various things. What we are doing here? We are again generating integers between 1 and 101. Since we want to include 100 also, between 1 and 101. And, in case, n is divisible by 2, or n is divisible by 3, or n is divisible by 5, then you increase the count by 1 and also s is equal to s plus n. And the initial value of s is 0, initial value of count is also 0.

And, then let us print how many such integers are there and then find the sum of all these. So, there are 74 such integers, which are divisible by 2,or 3 or 5 between 1 and 100 and sum of those integers is 3782.

(Refer Slide Time: 15:58)



```
74 3782
```

Use of continue

Example: How many integers are there between 1 and 100 (both inclusive) which are not divisible by 2, 3 and 5? Also print these numbers.

```
[22]: count = 0
      for n in range(1,100):
          if(n%2==0 or n%3==0 or n%5==0):
              continue
          count = count+1
          print(n, end=" ")
```

```
1 7 11 13 17 19 23 29 31 37 41 43 47 49 53 59 61 67 71 73 77 79 83 89 91 97
```

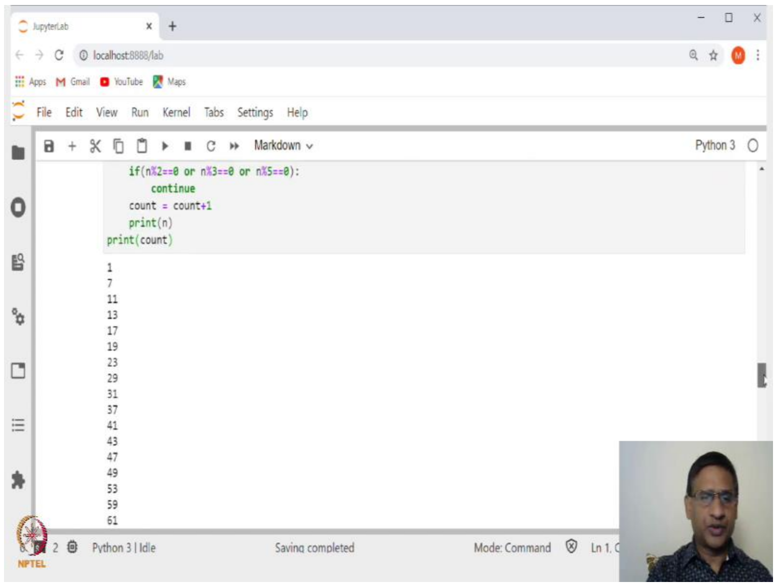
We have made use of break statement in order to break the loop in between. Sometimes, some particular iterate, you want to skip. So, for example, in this case we wanted these things to be added, but any integer which does not satisfy any of these things we do not want to add.

So, we can make use of continue statement. This is the same thing we want to do, the same task, but this time we want to add, how many integers are there between 1 and 100 again both included, which are not divisible by 2, 3 and 5. We want to add the remaining ones.

Of course, we could have done this using this kind of thing itself, but we will let us make use of continue. What we are going to do? So, count is equal to 0, which is simply want to count how many are there, but you can also add. And, then in case the number is divisible by 2 or 3 or 7 or 5, then you skip that.

So, continue will skip that iterate, when this condition is satisfied. That is what and if it and if it is not satisfied, then you add count by 1 and then print all of them ok. So, here it is printing everything in single line and that is where we are making use of end is equal to empty single quote or double quote, this simply say that do not break the line. If, I do not put this then the numbers will appear 1 in 1 at a time; 1 at a time.

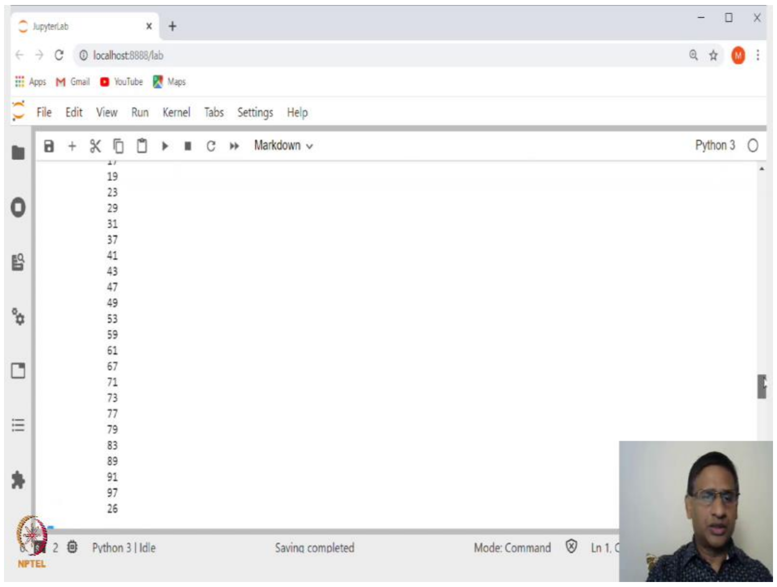
(Refer Slide Time: 17:53)



```
    if(n%2==0 or n%3==0 or n%5==0):
        continue
    count = count+1
    print(n)
print(count)
```

```
1
7
11
13
17
19
23
29
31
37
41
43
47
49
53
59
61
```

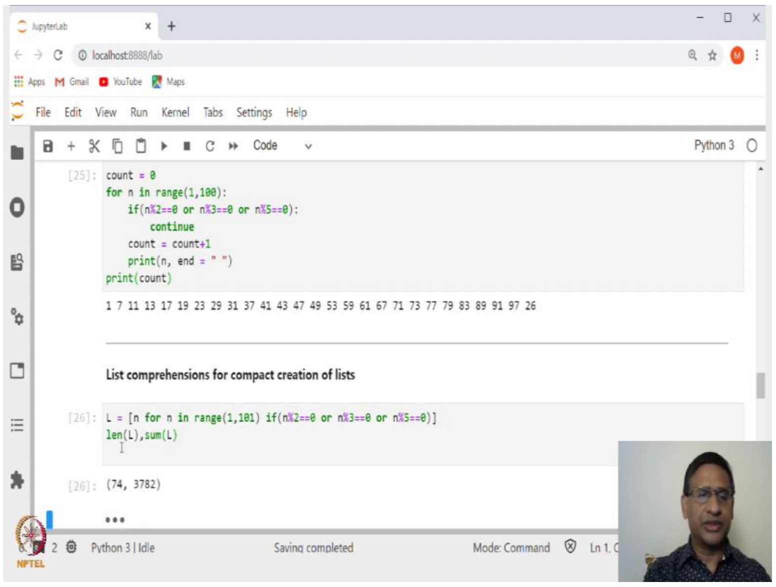
(Refer Slide Time: 18:01)



```
19
23
29
31
37
41
43
47
49
53
59
61
67
71
73
77
79
83
89
91
97
26
```

So, it is just printing all the integers and at the end let us say we want to also count how many are there. So you can print out side this let us say print count.

(Refer Slide Time: 18:22)



```
[25]: count = 0
      for n in range(1,100):
          if(n%2==0 or n%3==0 or n%5==0):
              continue
          count = count+1
          print(n, end = " ")
      print(count)
```

```
1 7 11 13 17 19 23 29 31 37 41 43 47 49 53 59 61 67 71 73 77 79 83 89 91 97 26
```

List comprehensions for compact creation of lists

```
[26]: L = [n for n in range(1,101) if(n%2==0 or n%3==0 or n%5==0)]
      len(L),sum(L)
```

```
[26]: (74, 3782)
```

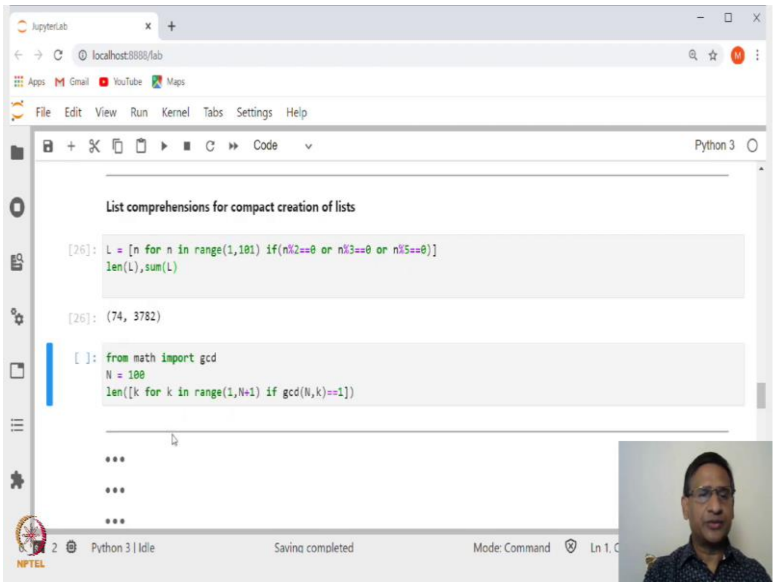
If I want all these numbers to appear in a single line, since there are not too many, then we can add here, what is called end is equal to empty double quote. The total is 26, the last one is the number is 26.

So, this is how you can make use of continue. Break and continue are very important concept that can be used in for loop in particular here. What does break do? Break will break the loop at intermediate state stage, whenever the condition is satisfied and continue will skip that particular step whenever that condition is satisfied.

Now many times, when you are writing the loop, you are writing everything in one line and one line. If you want to write everything in a single line, especially if you have a some kind of compact list to be generated, you can use it as follows. For example, let us say, we want to simply take all those integers between 1 and 101, which are divisible by 2 or 3 or 5.

So, we can simply write in the square bracket, the list we want to generate n, for n in range 1 to 100 and when do you take that n? If n is divisible by 2, 3 or 5. So, then we have generated a list and we can find length of this list, and sum of this list, by using function len and sum. So, this whole thing which we have done earlier could be done in single line. This is what we call list comprehensions and this will generate a compact list. This is a very nice feature, if you want to save time and space.

(Refer Slide Time: 20:34)

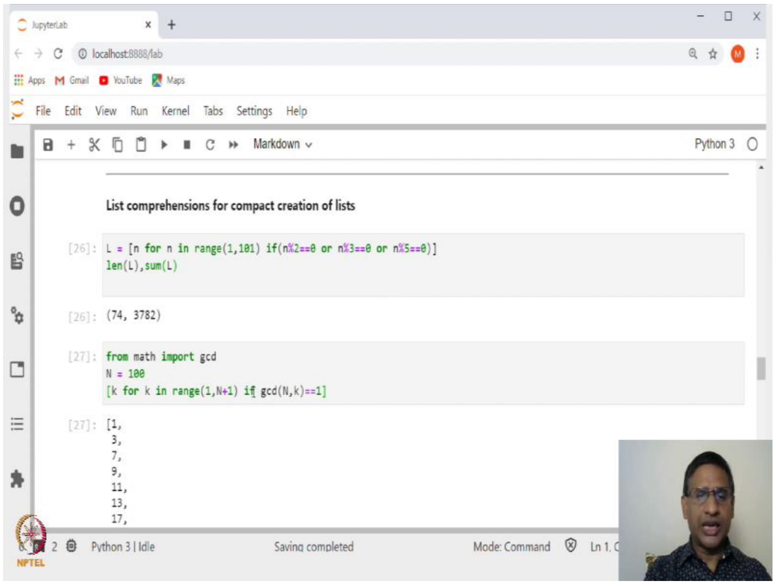


List comprehensions for compact creation of lists

```
[26]: L = [n for n in range(1,101) if(n%2==0 or n%3==0 or n%5==0)]
      len(L),sum(L)

[26]: (74, 3782)

[ ]:  from math import gcd
      N = 100
      len([k for k in range(1,N+1) if gcd(N,k)==1])
```

Similarly, if I want, let us say to print all those integers, which are co-prime to 100. All those integers, which are co-prime to 100, between 1 and100. Co-prime is same as saying, the gcd of that number with 100 should be equal to 1. Two integers with gcd 1, is known co-prime to each other. We already know that there is a function called gcd in math library or math module. So, let us import that function and capital N is 100.

(Refer Slide Time: 21:20)

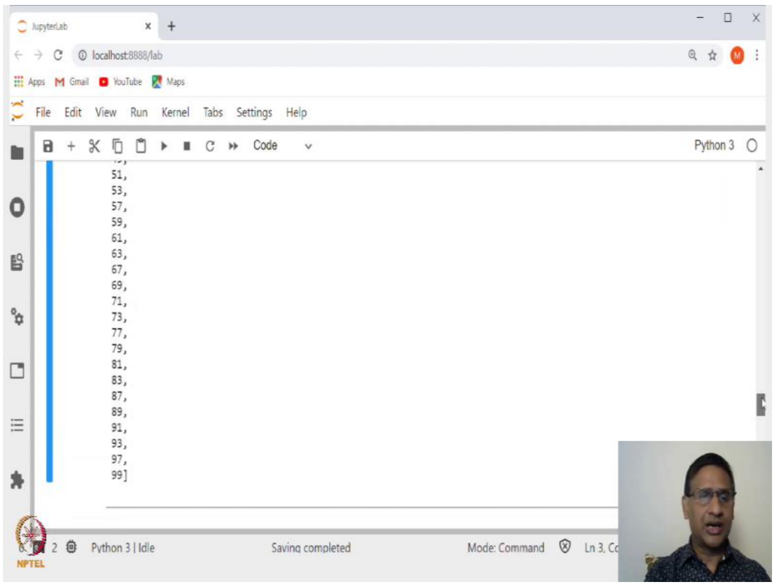


List comprehensions for compact creation of lists

```
[26]: L = [n for n in range(1,101) if(n%2==0 or n%3==0 or n%5==0)]
      len(L),sum(L)

[26]: (74, 3782)

[27]: from math import gcd
      N = 100
      [k for k in range(1,N+1) if gcd(N,k)==1]

[27]: [1,
       3,
       7,
       9,
       11,
       13,
       17,
```

Then we will generate let us say generate a list of all those integers k, for k in range 1 to 100, and when do we take that k, if gcd of N and k is 1. So, these are the integers.

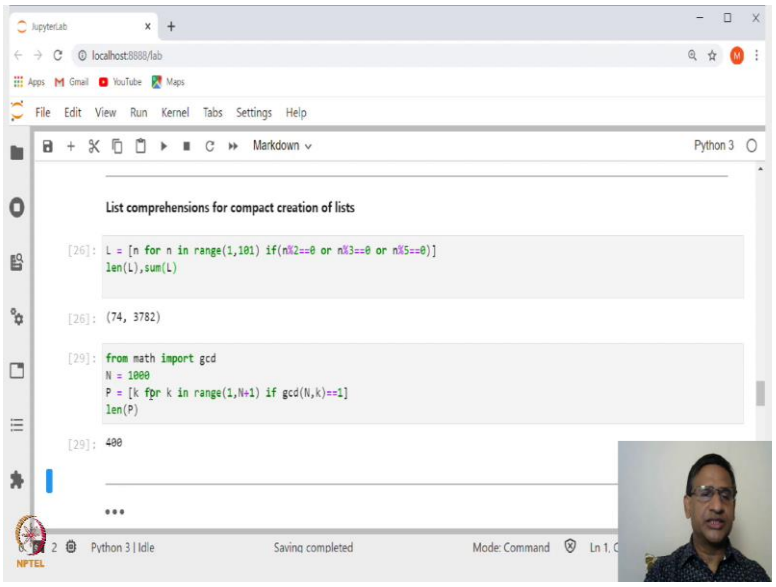
(Refer Slide Time: 21:33)



```
 51,
 53,
 57,
 59,
 61,
 63,
 67,
 69,
 71,
 73,
 77,
 79,
 81,
 83,
 87,
 89,
 91,
 93,
 97,
 99]
```

And, how many such integers are there? You can count by putting. So, let us say this list is P and then we will say len of P. So, that is 40.

(Refer Slide Time: 21:43)

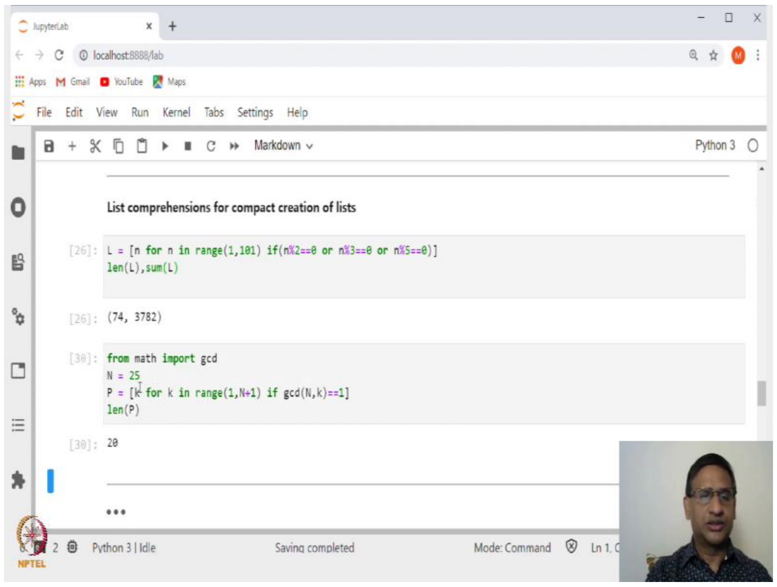


List comprehensions for compact creation of lists

```
[26]: L = [n for n in range(1,101) if(n%2==0 or n%3==0 or n%5==0)]
      len(L),sum(L)

[26]: (74, 3782)

[29]: from math import gcd
      N = 1000
      P = [k for k in range(1,N+1) if gcd(N,k)==1]
      len(P)

[29]: 400
```

How many such integers are there? You can count by putting, let us say this list is P and then we will say len of P. So, that is 40. There are 40 integers between 1 and 100, which are co-prime to 100.

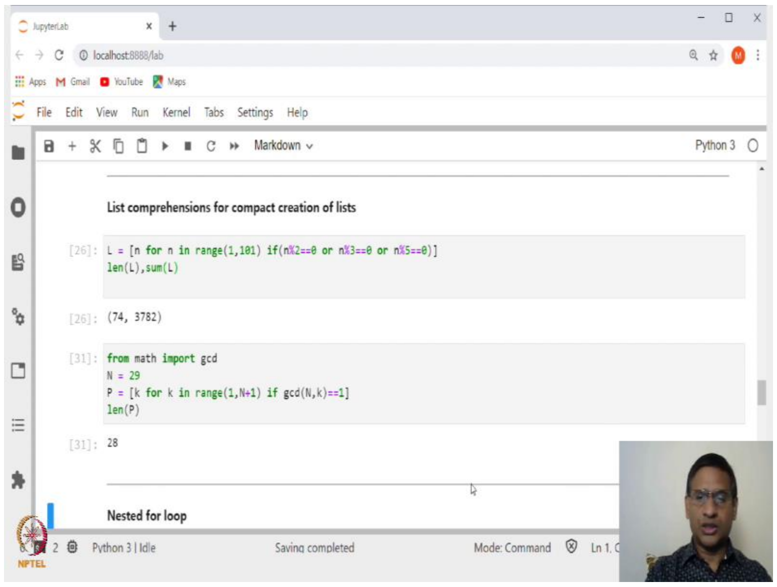
(Refer Slide Time: 22:06)


List comprehensions for compact creation of lists

```
[26]: L = [n for n in range(1,101) if(n%2==0 or n%3==0 or n%5==0)]
      len(L),sum(L)

[26]: (74, 3782)

[30]: from math import gcd
      N = 25
      P = [k for k in range(1,N+1) if gcd(N,k)==1]
      len(P)

[30]: 20
```

If, I put for example, let us say 25 there are 20 integers ok.

(Refer Slide Time: 22:16)


List comprehensions for compact creation of lists

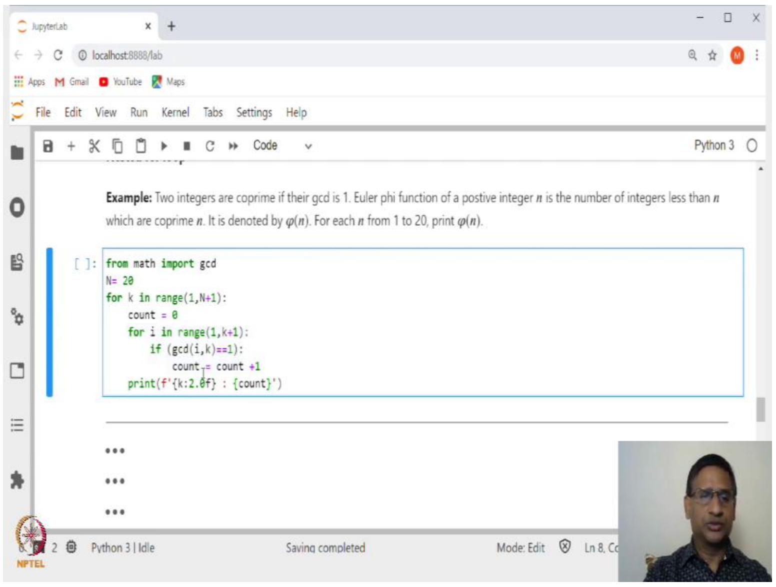
```
[26]: L = [n for n in range(1,101) if(n%2==0 or n%3==0 or n%5==0)]
      len(L),sum(L)

[26]: (74, 3782)

[31]: from math import gcd
      N = 29
      P = [k for k in range(1,N+1) if gcd(N,k)==1]
      len(P)

[31]: 28
```

Nested for loop

If, I have, let us say some prime, for example, 29 and what are integers which are co-prime to 29. So, it will give you 1 to 28. There will be 28, that's correct.

Let us look at another example. Many times you want to use for loop inside for loop. This is what is called nested for loop. We have seen, we can use if inside if so, nested ifs, we have looked at. We can also use nested for loops.

(Refer Slide Time: 22:44)


Example: Two integers are coprime if their gcd is 1. Euler phi function of a postive integer n is the number of integers less than n which are coprime n. It is denoted by $\varphi(n)$. For each n from 1 to 20, print $\varphi(n)$.

```
[ ]: from math import gcd
     N= 20
     for k in range(1,N+1):
         count = 0
         for i in range(1,k+1):
             if (gcd(i,k)==1):
                 count = count +1
         print(f'{k:2.0f} : {count}')
```

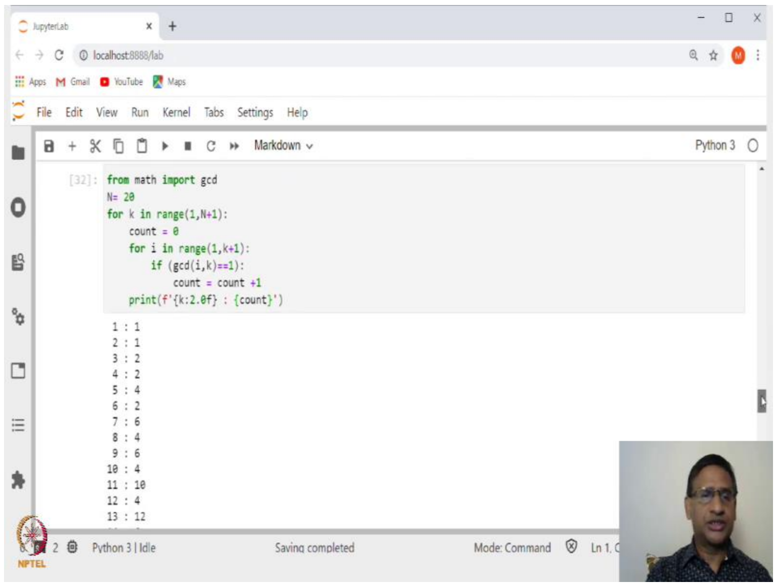
Let us look at an example, in earlier example we saw, how to find the number of integers, which are co-prime to a given integer. Now, let us say, if you look at the number of integers which are co-prime between 1 and n, these are called Euler-phi function of n.

So, what we want to do? We want to print Euler phi function for each of integer between 1 and 20. So, how many integers are there which are co-prime to 1, 2, 3 and up to 20? So one loop will be to generate how many integers are there which are co prime to particular integer k between1 and 20 and other loop will run, that is, outer loop which will run from 1 to 20.

So, this is a nested loop. Again let us import gcd, though we have already imported, and from 1 to 20, is what we want to print. This for loop will be for each integer between 1 to 20 we want to generate how many co-prime integers are there.

We could have also used this here list comprehension, the one which we did and then print for each one.

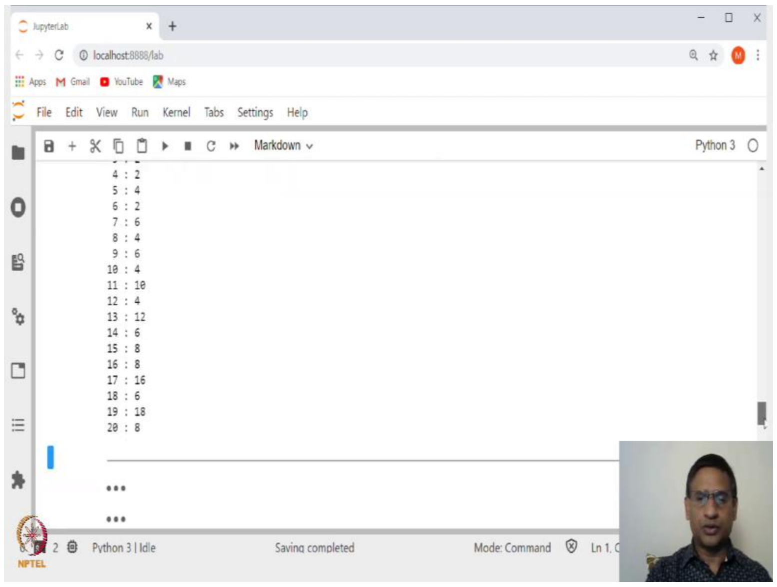
(Refer Slide Time: 24:26)



```
[32]: from math import gcd
      N= 20
      for k in range(1,N+1):
          count = 0
          for i in range(1,k+1):
              if (gcd(i,k)==1):
                  count = count +1
          print(f'{k:2.0f} : {count}')

 1 : 1
 2 : 1
 3 : 2
 4 : 2
 5 : 4
 6 : 2
 7 : 6
 8 : 4
 9 : 6
10 : 4
11 : 10
12 : 4
13 : 12
```

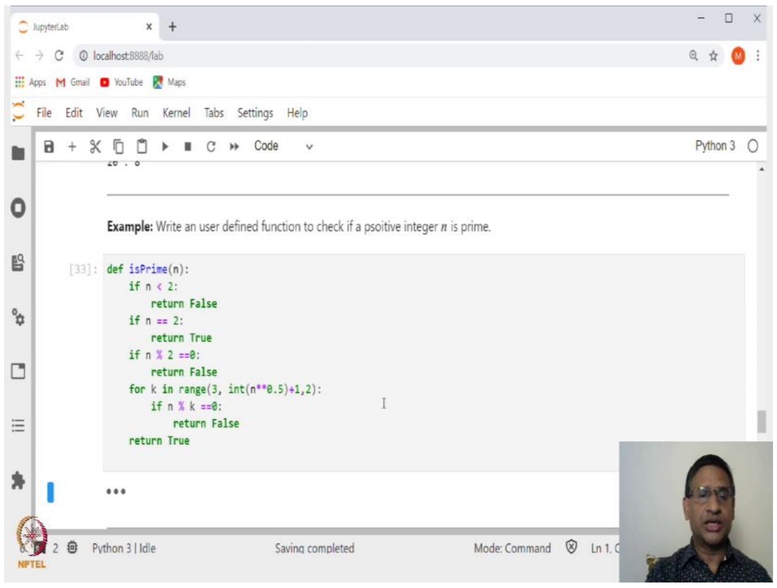
So, it says that 1phi of 1 is 1, phi of 2 is 1, phi of 3 is 2, phi of 4 is 2 and so on. So, that is a list and we have used nested for loop, for loop inside a for loop.

(Refer Slide Time: 24:42)



```
 4 : 2
 5 : 4
 6 : 2
 7 : 6
 8 : 4
 9 : 6
10 : 4
11 : 10
12 : 4
13 : 12
14 : 6
15 : 8
16 : 8
17 : 16
18 : 6
19 : 18
20 : 8
```

(Refer Slide Time: 24:51)



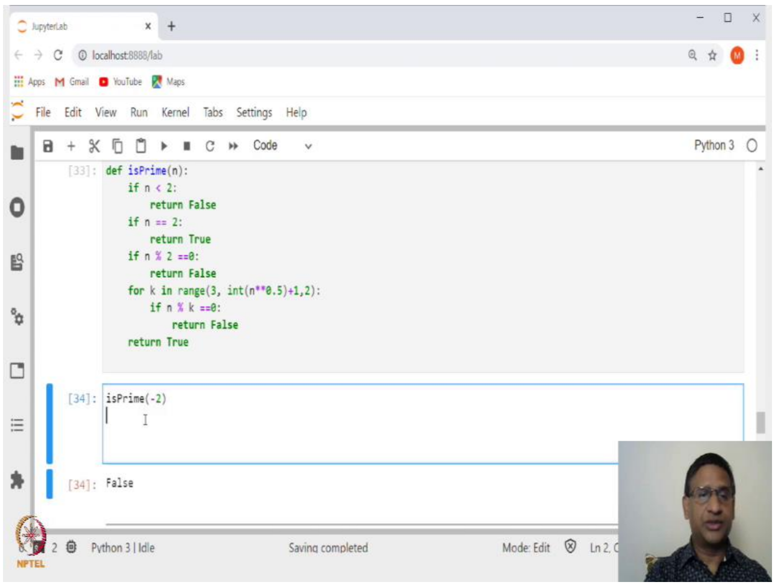
Example: Write an user defined function to check if a psoitive integer *n* is prime.

```
[33]: 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
```

Let us look at another example. Suppose we want to write an user defined function to check if a positive integer is prime. How do we do that? So, let me call this function as isPrime, the prime starts from 2. So, if anything which is less than 2 you return false, if n is equal to 2, of course, it is true statement. If n is divisible by 2, again it is not a prime. So, you just simply return False. Otherwise you check k for k between 3 and, this is actually you need not go all the way up to n. It is good enough, if n has a divisor the divisor will be less than equal to square root of n. So, if you can check up to square root of n, If you cannot find up to a square root of n, then you cannot find it later. That is why this runs from3 to square root of n, n to the power 0.5. If n, we take as, let us say some square integer, then square root will be an integer, but otherwise it is not an integer and that is why we are making this as integer and then you have to go up to exactly up to a square root n.

Then you need not check the even integers, that is why start from 3 and increase by 2. In case n is divisible by 0, you return False. If none of these things are True then you print True, return true. So, that is the function to check, if a given integer is prime or not. Now, let us make use of this. So, suppose we want to check various things.

(Refer Slide Time: 26:55)



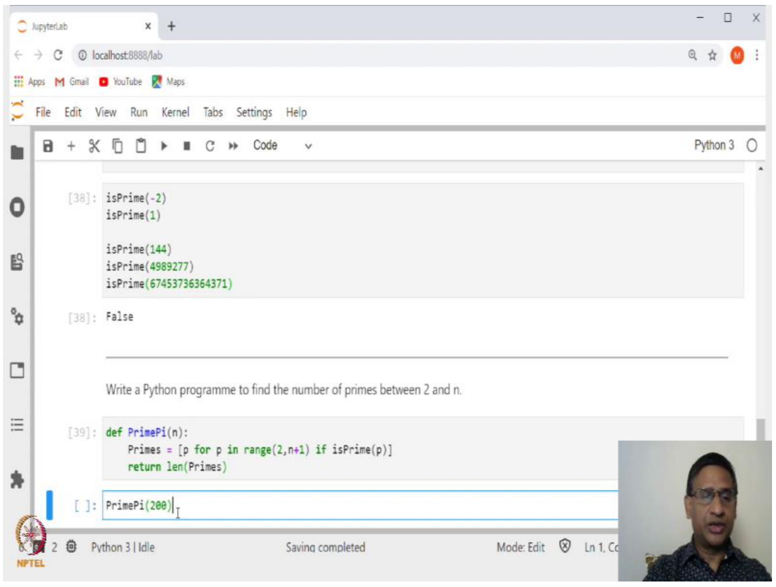
```
[33]: def isPrime(n):
          if n < 2:
              return False
          if n == 2:
              return True
          if n % 2 ==0:
              return False
          for k in range(3, int(n**0.5)+1,2):
              if n % k ==0:
                  return False
          return True
```

```
[34]: isPrime(-2)
```

```
[34]: False
```

So, if I take whether minus 2 is a prime or not. Obviously, this is less than 2, therefore, it is not a prime. If I want to check whether 1 is prime or not. So, again this should be false.

(Refer Slide Time: 27:10)



```
[38]: isPrime(-2)
      isPrime(1)

      isPrime(144)
      isPrime(4989277)
      isPrime(67453736364371)
```

```
[38]: False
```

Write a Python programme to find the number of primes between 2 and n.

```
[39]: def PrimePi(n):
          Primes = [p for p in range(2,n+1) if isPrime(p)]
          return len(Primes)
```

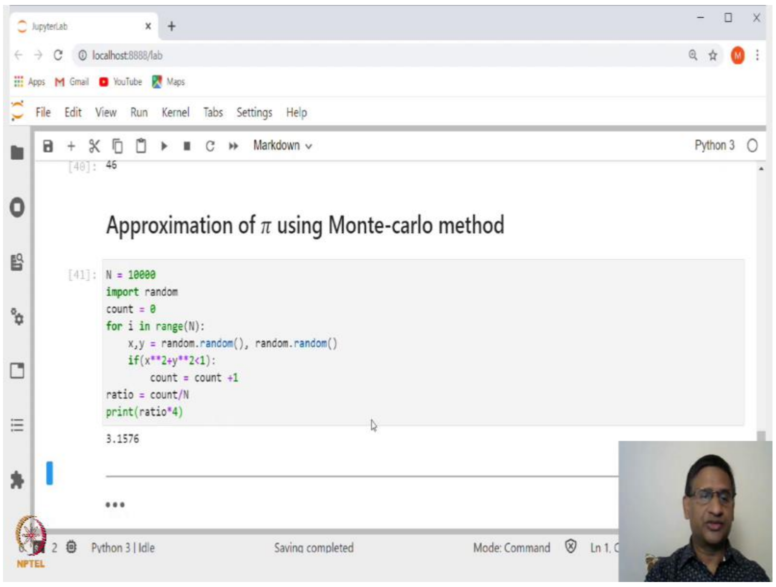
```
[ ]: PrimePi(200)
```

If, I want to check, let us say 144 is prime or not the answer is False.

 If I want to check, let us say 4989277 is prime number the answer is true. So, if I give any arbitrary number isPrime and some arbitrary number, let us say it ends with 1, the answer is False.

Now suppose we want to write a python program to find the number of primes between 2 and n, for particular n. So, how will you do that? Already, we have a function isPrime to check the whether prime or not. Again we will make use of list comprehension. So, I am just giving the name of this function as PrimePi and the argument is n. Generate all these primes between 2 to n. So, p for p in range 2 to n plus 1, when do you take that? n if it gives you value true, if isPrime value is equal to true. So, if this is true then it will take p. It makes a list of Primes and then return the length of this. So, that is how you can find number of Primes. Suppose we want to find out how many primes are there between 2 and 100 ? The answer is 46.

(Refer Slide Time: 28:40)



```
[40]: 46
```

Approximation of π using Monte-carlo method

```
[41]: N = 10000
      import random
      count = 0
      for i in range(N):
          x,y = random.random(), random.random()
          if(x**2+y**2<1):
              count = count +1
      ratio = count/N
      print(ratio*4)

      3.1576
```

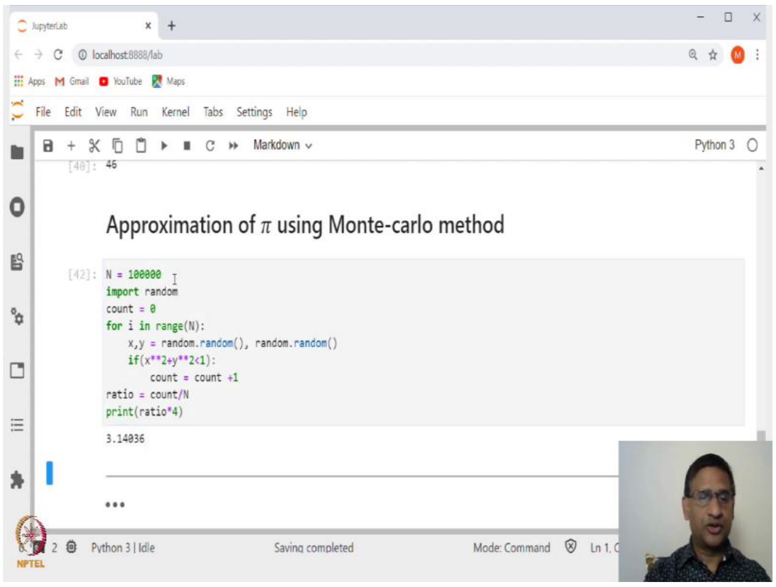
Let me give you another example. Let us say, we want to approximate pi using, what is known as Monte Carlo method. This method is very simple, the way it works is that, for example, you take a unit square. And inside that make a circle of radius, let us say, 1. This circle is, let us say centered at (0, 0). And, then what you do is, you generate some large number of random points. So, each point will have coordinate between minus 1 and 1. And, then count how many of these points are inside this circle and how many points are outside the circle? If, it is a unit square it is area is 1, however, the area of the unit circle will be will be pi.

So, the probability of a point inside the circle, you can find out and that is how you can approximate the value of pi. This is what is known as Monte Carlo method.

Instead of generating between inside unit circle, we can generate random number between 0 and 1. So, each point will lie in the first quadrant. We are taking the first quadrant of the circle and then you take one-fourth of the of the unit square that is in the first quadrant. So, what we do is, let us say we generate 10,000 points.   Again import the random, and let us say start the count equal to 0. So, we will be counting how many how many points are there which are inside the circle?

So, we are generating each coordinate x and y as random dot random and random dot random. So, x and y are random points between 0 and 1. In case the x square plus y square is less than 1, then you say that it is lying inside the circle that will increase the count by 1, and then look at the ratio. The total number of points which are inside the circle, inside the first quadrant divided by the total number of points, which was 10000 and then multiply that ratio by 4, you will be getting that number which is approximation of pi.

(Refer Slide Time: 31:25)



[40]: 46

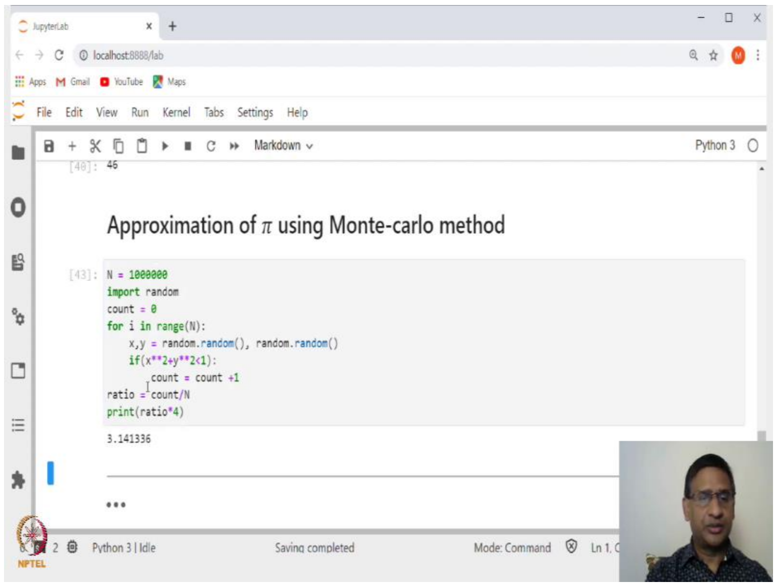
Approximation of π using Monte-carlo method

[42]:
```
N = 100000
import random
count = 0
for i in range(N):
    x,y = random.random(), random.random()
    if(x**2+y**2<1):
        count = count +1
ratio = count/N
print(ratio*4)
```

```
3.14036
```

So, if I run this, this is 3.1576, if I increase the number of points let us say, then you will get closer to pi.

(Refer Slide Time: 31:30)



```
[40]: 46
```

Approximation of π using Monte-carlo method

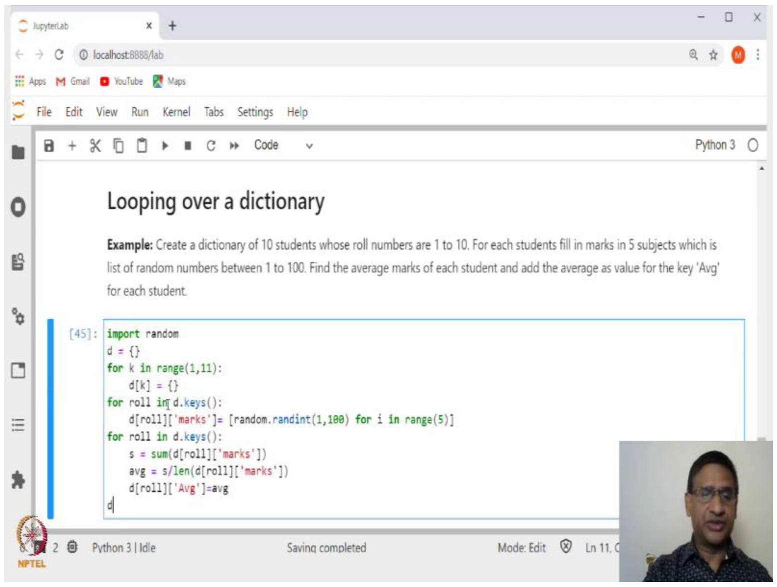
```
[43]: N = 1000000
import random
count = 0
for i in range(N):
    x,y = random.random(), random.random()
    if(x**2+y**2<1):
        count = count +1
ratio = count/N
print(ratio*4)

3.141336
```

```
2    Python 3 | Idle    Saving completed    Mode: Command    Ln 1,
```

If, I further increase this, it may take more time, but you will get much closer to pi.

So, if you will generate large number of such points inside unit square and count how many are there inside unit circle. Then and take the ratio multiply that by 4, that gives you, a way of approximating pi using Monte Carlo method. There are other ways of looking at it.

(Refer Slide Time: 31:56)



Looping over a dictionary

Example: Create a dictionary of 10 students whose roll numbers are 1 to 10. For each students fill in marks in 5 subjects which is list of random numbers between 1 to 100. Find the average marks of each student and add the average as value for the key 'Avg' for each student.

```
[45]: import random
d = {}
for k in range(1,11):
    d[k] = {}
for roll in d.keys():
    d[roll]['marks']= [random.randint(1,100) for i in range(5)]
for roll in d.keys():
    s = sum(d[roll]['marks'])
    avg = s/len(d[roll]['marks'])
    d[roll]['Avg']=avg
d
```

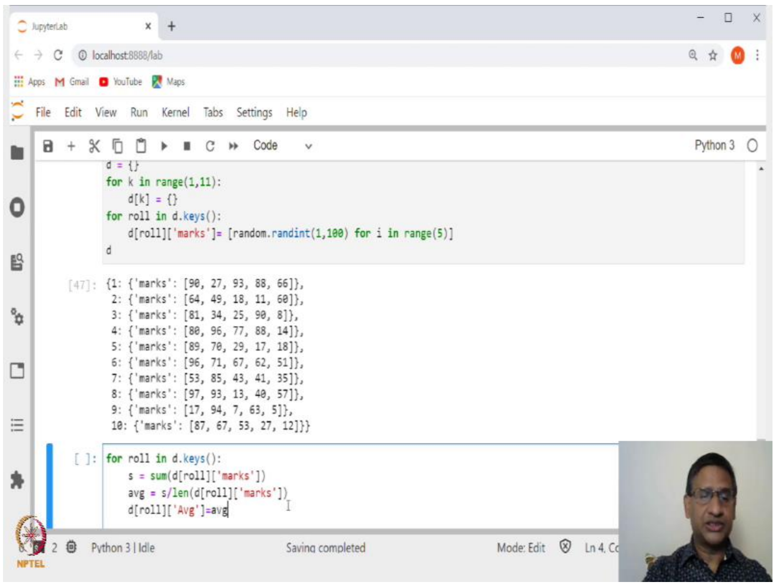
```
2    Python 3 | Idle    Saving completed    Mode: Edit    Ln 11,
```

Similarly, I said in the beginning the for variable inside X and that X can be dictionary as well. So, let us take an example of this. Suppose you want to create a dictionary of 10 students, with roll number 1 to 10, and for each student you have marks in 5 subjects, which is random number between 1 and 100. Then find the average of each of students and then add that average as another key to this dictionary.

First let us import random, then what we will do? Let me just show you one by one, it will be easier to understand. First we generate an empty dictionary. Next we generate roll number and roll number is again a dictionary, which is empty dictionary. The value of that is empty dictionary. Then what do we do?

Next we will generate random the marks. So, marks is a key inside this dictionary and that marks is a list of 5 random integers between 1 and 100.

(Refer Slide Time: 33:30)



```
d = {}
for k in range(1,11):
    d[k] = {}
for roll in d.keys():
    d[roll]['marks']= [random.randint(1,100) for i in range(5)]
d
```

```
[47]: {1: {'marks': [90, 27, 93, 88, 66]},
       2: {'marks': [64, 49, 18, 11, 60]},
       3: {'marks': [81, 34, 25, 90, 8]},
       4: {'marks': [80, 96, 77, 88, 14]},
       5: {'marks': [89, 70, 29, 17, 18]},
       6: {'marks': [96, 71, 67, 62, 51]},
       7: {'marks': [53, 85, 43, 41, 35]},
       8: {'marks': [97, 93, 13, 40, 57]},
       9: {'marks': [17, 94, 7, 63, 5]},
       10: {'marks': [87, 67, 53, 27, 12]}}
```

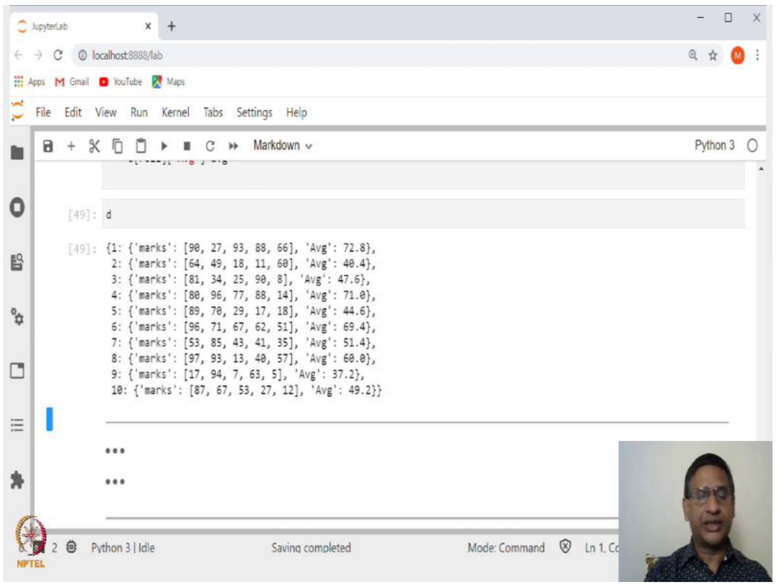
```
[ ]: for roll in d.keys():
         s = sum(d[roll]['marks'])
         avg = s/len(d[roll]['marks'])
         d[roll]['Avg']=avg
```

Let us generate this. Now, suppose if I say, what is d, you will see that we have generated a dictionary of 10 keys and the keys are roll number 1 to 10. Now for each of roll number the value is again a dictionary. So, this is a nested dictionary, and the marks is a list of 5 integers.

Now, what we want to do? We want to find the average of each of these marks and add this to average. So, what we will do? We can apply for loop inside dictionary. So, for roll in d dot keys, so, d dot keys will give me 1 2 3 up to 10.

So, for each of this now, you look at the sum of the marks. So, how do I access the marks? It is d of roll and the marks of that. So, these are the list of marks and then take the sum, that is the sum of the marks of each student and then take the average, average is s divided by the length of the marks and then you add inside this roll number. You create another key, which is called average and then add that value to the average and let us run this.

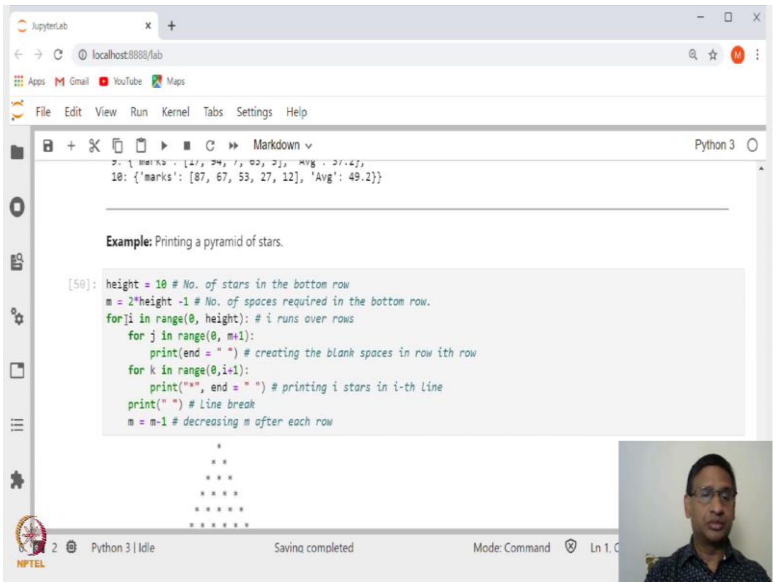
(Refer Slide Time: 35:02)



```
[49]: d

[49]: {1: {'marks': [90, 27, 93, 88, 66], 'Avg': 72.8},
       2: {'marks': [64, 49, 18, 11, 60], 'Avg': 40.4},
       3: {'marks': [81, 34, 25, 90, 8], 'Avg': 47.6},
       4: {'marks': [80, 96, 77, 88, 14], 'Avg': 71.0},
       5: {'marks': [89, 70, 29, 17, 18], 'Avg': 44.6},
       6: {'marks': [96, 71, 67, 62, 51], 'Avg': 69.4},
       7: {'marks': [53, 85, 43, 41, 35], 'Avg': 51.4},
       8: {'marks': [97, 93, 13, 40, 57], 'Avg': 60.0},
       9: {'marks': [17, 94, 7, 63, 5], 'Avg': 37.2},
       10: {'marks': [87, 67, 53, 27, 12], 'Avg': 49.2}}
```

Now if I ask for what is d? You will see that you have first roll number, these are the marks and it is average is added. So, this is one example in which we are iterating for loop over a dictionary.

(Refer Slide Time: 35:18)



```
 10: {'marks': [87, 67, 53, 27, 12], 'Avg': 49.2}}
```

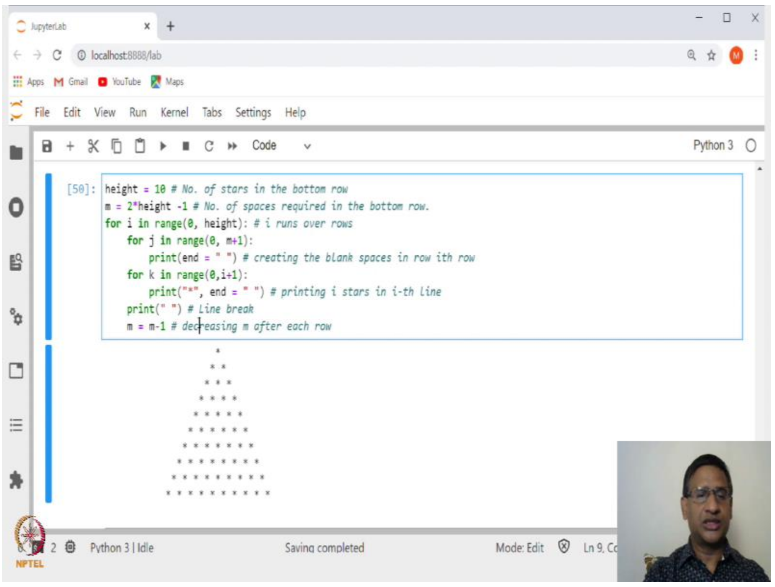
Example: Printing a pyramid of stars.

```
[50]: height = 10 # No. of stars in the bottom row
      m = 2*height -1 # No. of spaces required in the bottom row.
      for i in range(0, height): # i runs over rows
          for j in range(0, m+1):
              print(end = " ") # creating the blank spaces in row ith row
          for k in range(0,i+1):
              print("*", end = " ") # printing i stars in i-th line
          print(" ") # Line break
          m = m-1 # decreasing m after each row

          *
         * *
        * * *
       * * * *
      * * * * *
     * * * * * *
```

Let me just give you one more example, suppose we want to print a pyramid of stars. This is an example of nested for loop.   Here actually we are using three for loops.

So, the way it works is as follows. So, in this case we are taking height to be 10 height is the number of stars you want in this pyramid.

(Refer Slide Time: 35:40)



```
[50]: height = 10 # No. of stars in the bottom row
      m = 2*height -1 # No. of spaces required in the bottom row.
      for i in range(0, height): # i runs over rows
          for j in range(0, m+1):
              print(end = " ") # creating the blank spaces in row ith row
          for k in range(0,i+1):
              print("*", end = " ") # printing i stars in i-th line
          print(" ") # Line break
          m = m-1 # decreasing m after each row
```

```
         *
        * *
       * * *
      * * * *
     * * * * *
    * * * * * *
   * * * * * * *
  * * * * * * * *
 * * * * * * * * *
* * * * * * * * * *
```

Let me just give you one more example, suppose we want to print a pyramid of stars. This is an example of nested for loop.   Here actually we are using three for loops.

So, the way it works is, as follows. So, in this case, we are taking height to be 10, height is the number of rows you want in this pyramid. We want that as 10 and then what we are doing is, we look at how many spaces it will require? So, this is the number of spaces which will be required is 2 times height minus 1.

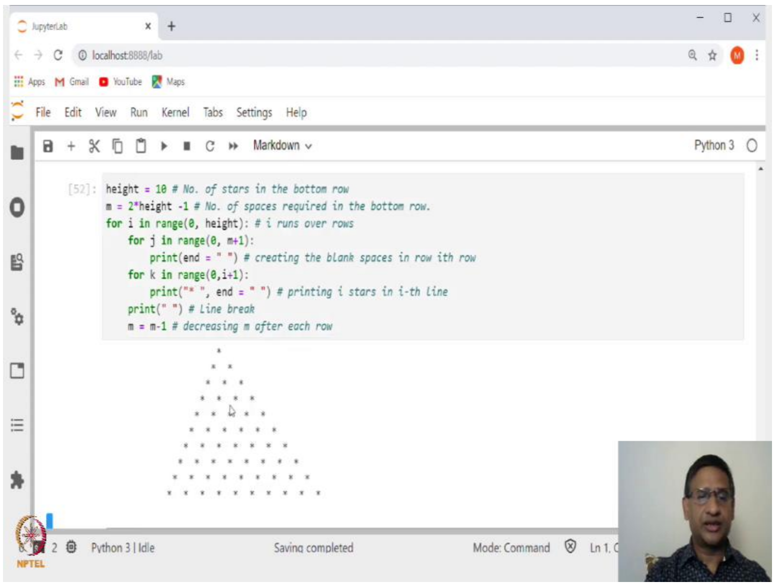
That means, total 19 spaces you are going to require and then for each row you have for loop which is this one. This is for each row. Total number of row is the total height which is 0 to10 in this case. And this is a loop, which will print actually the extra spaces.

For example, in the first row you will have so many extra spaces. And, that extra spaces is between 0 to m plus 1 where m is 2 times height minus 1. For example, if I start with this j is equal to 0, then that in the first row the number empty spaces, that it should print first is going to be, this is 20 minus 1 which 19 plus 1, that is 20. So, it will print 19 empty spaces. And, then the next loop is, you are printing star and so, this this is k is varying between 0 to i plus

1. As you increase the row, the number of stars will keep increasing by 1 and that is what is happening? Then you print empty line, that is a new line, this is for the line break and then you decrease m by 1.

Now if you run, this you will get this.

(Refer Slide Time: 37:41)

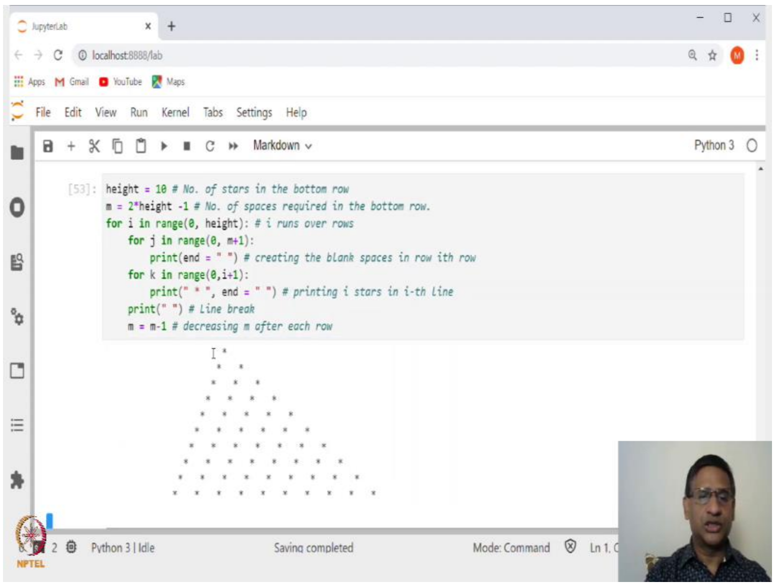


```
[52]: height = 10 # No. of stars in the bottom row
      m = 2*height -1 # No. of spaces required in the bottom row.
      for i in range(0, height): # i runs over rows
          for j in range(0, m+1):
              print(end = " ") # creating the blank spaces in row ith row
          for k in range(0,i+1):
              print("* ", end = " ") # printing i stars in i-th line
          print(" ") # Line break
          m = m-1 # decreasing m after each row
```

```
          *
         * *
        * * *
       * * * *
      * * * * *
     * * * * * *
    * * * * * * *
   * * * * * * * *
  * * * * * * * * *
 * * * * * * * * * *
```

If I give you for example; star and some space here you will get one extra space.

(Refer Slide Time: 37:45)



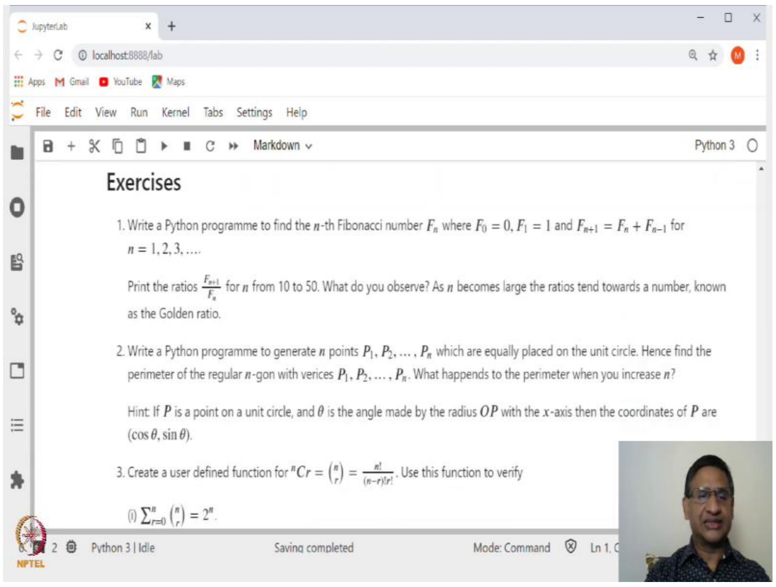
```
[53]: height = 10 # No. of stars in the bottom row
      m = 2*height -1 # No. of spaces required in the bottom row.
      for i in range(0, height): # i runs over rows
          for j in range(0, m+1):
              print(end = " ") # creating the blank spaces in row ith row
          for k in range(0,i+1):
              print(" * ", end = " ") # printing i stars in i-th line
          print(" ") # Line break
          m = m-1 # decreasing m after each row
```

```
                    *
                  *   *
                *   *   *
              *   *   *   *
            *   *   *   *   *
          *   *   *   *   *   *
        *   *   *   *   *   *   *
      *   *   *   *   *   *   *   *
    *   *   *   *   *   *   *   *   *
  *   *   *   *   *   *   *   *   *   *
```

If, I give one space in the beginning also this is what is going to happen.

And, you can generate this pyramid downwards you can generate a diamond kind of thing also. So, this is a very nice example of nested for loop.

(Refer Slide Time: 38:02)



Exercises

1. Write a Python programme to find the n-th Fibonacci number F_n where $F_0 = 0, F_1 = 1$ and $F_{n+1} = F_n + F_{n-1}$ for $n = 1, 2, 3, \ldots$.

   Print the ratios $\frac{F_{n+1}}{F_n}$ for n from 10 to 50. What do you observe? As n becomes large the ratios tend towards a number, known as the Golden ratio.

2. Write a Python programme to generate n points $P_1, P_2, \ldots, P_n$ which are equally placed on the unit circle. Hence find the perimeter of the regular n-gon with verices $P_1, P_2, \ldots, P_n$. What happends to the perimeter when you increase n?

   Hint: If P is a point on a unit circle, and θ is the angle made by the radius OP with the x-axis then the coordinates of P are $(\cos\theta, \sin\theta)$.

3. Create a user defined function for ${}^nCr = \binom{n}{r} = \frac{n!}{(n-r)!r!}$. Use this function to verify

   (i) $\sum_{r=0}^{n} \binom{n}{r} = 2^n$.

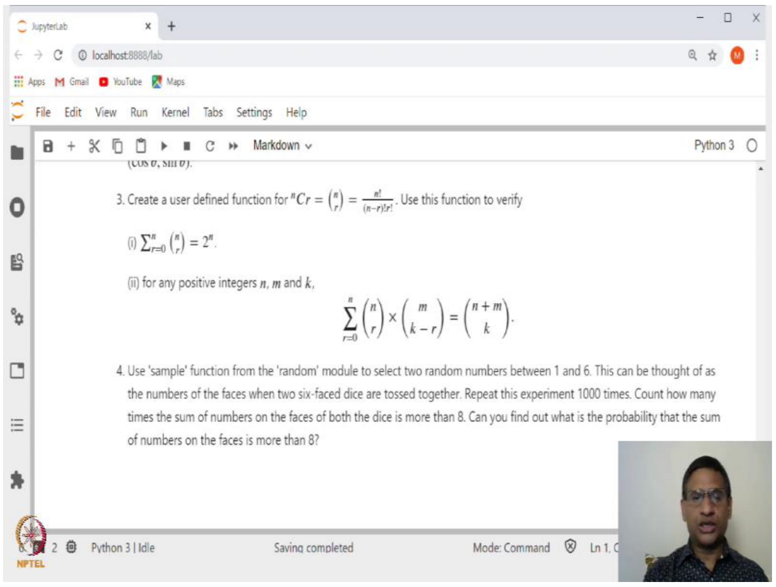
At the end let me leave you with some simple exercises.

The first exercise is going to be write a Python programme to find, n th Fibonacci number. The n th Fibonacci number is defined is given by recurrence F n plus 1 is equal to F n plus F n minus 1, and where n start with 2 initial values, F 0 is let us say 0 F 1 is 1. So, this is what we call Fibonacci sequence. And, in case you take the ratio or F n plus 1 divide by F n and increase n to the large value, then that ratio actually converges to very famous constant called the golden ratio, which has a lot of applications in nature.

So, what I want to do is, you generate the ratio of F n divide by F n plus 1 between let us say 1 and 50 for n is equal to 1 to 50, you generate this. So, first create an user defined function for Fibonacci sequence, generating, Fibonacci number and then print these golden ratio. And, you will see that this golden ratio, this number as you increase n it will converge to a particular value, which is what is called golden ratio.

Next exercise is write a Python programme to generate n points let us say, P1 to Pn which are equally placed on unit circle. So, on unit circle you generate n equally placed points. Any points on the unit circle is of the form cos theta sine theta, where theta is the angle, line joining the point and the origin makes with positive x axis. Hence you find the perimeter of the regular n-gon with vertices P1 to Pn. And, as you increase this n you will see that that regular n-gon will become like a perimeter of a circle, and so, that this is another way of approximating the value of pi.

(Refer Slide Time: 40:05)



$(\cos\theta, \sin\theta)$.

3. Create a user defined function for $^nCr = \binom{n}{r} = \frac{n!}{(n-r)!r!}$. Use this function to verify

(i) $\sum_{r=0}^{n} \binom{n}{r} = 2^n$.

(ii) for any positive integers n, m and k,

$$\sum_{r=0}^{n} \binom{n}{r} \times \binom{m}{k-r} = \binom{n+m}{k}.$$

4. Use 'sample' function from the 'random' module to select two random numbers between 1 and 6. This can be thought of as the numbers of the faces when two six-faced dice are tossed together. Repeat this experiment 1000 times. Count how many times the sum of numbers on the faces of both the dice is more than 8. Can you find out what is the probability that the sum of numbers on the faces is more than 8?

The next is verification of these two identity, which is sum of n c r, r going from 0 to n is 2 to the power n.

And, the next one is sum of n c r into m c k minus r, r varying from 0 to n is equal to m plus n divided by n plus n c k.

And, the one last one is use sample function from random module. So, this sample function will generate a random sample. So, if you have a list of 20 numbers and if you want to take first 10 of them, you can make use of this random function. So, use this random function to generate a random number between 1 and 6. So, it is like a throwing a dice, the random number which you get is the value on the face.

So, this this can be thought of as a number of faces when two six-face dice are tossed together. Repeat this experiment 100 times. And, then count how many times the sum of the number of the faces of the both the dice are more than 8.

So, two dice throwing together and that is why you are generating two random numbers between 1 and 6. Then repeat this experiment 1000 times, and count how many times, you have got the sum of the numbers more than 8. And, hence you can also find out the probability that the sum of the numbers on the two faces is more than 8.

So, these are the simple exercises of course, we will be posting you the solution of these things, but they are actually fairly simple.

I am sure all of you will be able to write program for this. Also spend more time on working on these things because that will also teach you for loop and nested for loop etc.

Next time we will be looking at while loop. In case of for loop you know how many times in the beginning itself, it should run except that when we are using break statement. And, in while loop that is not the case while certain conditions are satisfied you can run this loop.

So, thank you very much. I will see you in the next class.