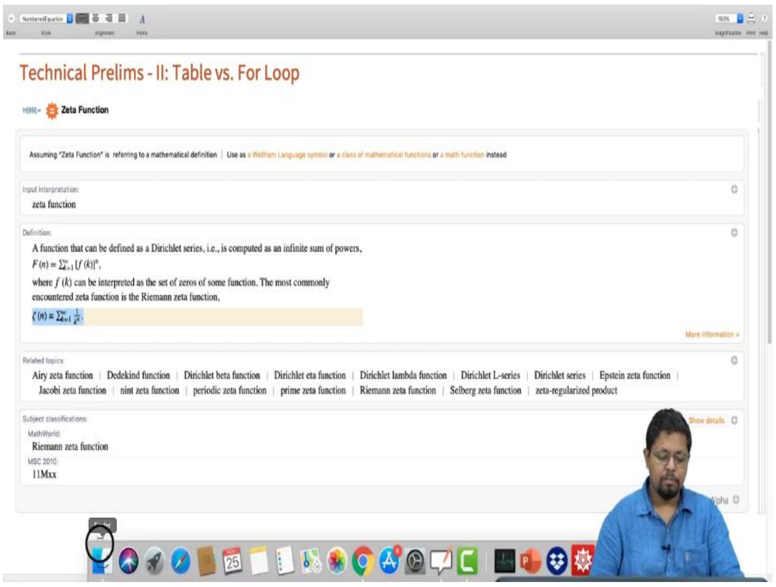
Physics through Computational Thinking
Professor Auditya Sharma & Professor Ambar Jain
Department of Physics
Indian Institute of Science Education and Research, Bhopal
Lecture 11
Technical Prelims 2

(Refer Slide Time: 0:26)



Technical Prelims - II: Table vs. For Loop

In[85]:= Zeta Function

Assuming "Zeta Function" is referring to a mathematical definition | Use as a Wolfram Language symbol or a class of mathematical functions or a math function instead

Input interpretation:
zeta function

Definition:
A function that can be defined as a Dirichlet series, i.e., is computed as an infinite sum of powers,
$F(n) = \sum_{k=1}^{\infty} [f(k)]^n$,
where $f(k)$ can be interpreted as the set of zeros of some function. The most commonly encountered zeta function is the Riemann zeta function,
$\zeta(n) = \sum_{k=1}^{\infty} \frac{1}{k^n}$.

More information »

Related topics:
Airy zeta function | Dedekind function | Dirichlet beta function | Dirichlet eta function | Dirichlet lambda function | Dirichlet L-series | Dirichlet series | Epstein zeta function | Jacobi zeta function | nint zeta function | periodic zeta function | prime zeta function | Riemann zeta function | Selberg zeta function | zeta-regularized product

Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

Okay, welcome back and today we will talk about technical prelims two. This is a technical video, so it is all going to be about the technicalities of Mathematica commands and some technicalities of programming, we will not talk too much about physics and mathematics here.

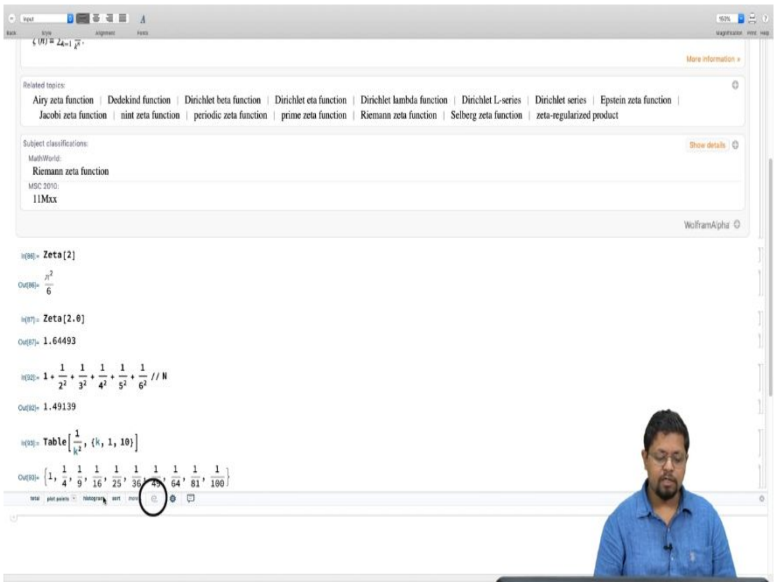
Today we will talk particularly about For loop or how to write a For loop in Mathematica and we will contrast and compare with the Table command. You are already familiar with a Table command, if you want to generate a series of data you can use the Table command. So, Table command sounds a lot like a loop, something a loop will do for you. But For loop is something that is different kind of construct. And today we will contrast and understand the difference between Table and For loop. So let us go ahead and get started with it.

To take an example, I will use something known as the Riemann zeta function. So, I will search about Riemann zeta function and notice the symbol over here. Let me type this out and show it to you again. If I type equal to once and then again I type an equal to, this becomes a equal to inside a star symbol like this, and this becomes a natural language, Mathematica's natural language tool where I can type something and Mathematica will search the internet in its own knowledge database and tell me about what that is.

So, here I have searched about the zeta function or Riemann zeta function. Riemann zeta function is a famous function and it is giving an interpretation here assuming the zeta function is referring to a mathematical definition as giving a disambiguation over here, it says, it gives you a bunch of information about the zeta function.

If you are not familiar with the Riemann zeta function, do not worry about it. It is actually simply given by this series. So, Zeta[n] is simply given by this series, it is an infinite sum of numbers. It is a sum over $1/k^n$, where k is from 1 to infinity. So, Zeta[n] is $\sum_{k=1}^{\infty} 1/k^n$.

(Refer Slide Time: 2:35)


Related topics:
Airy zeta function | Dedekind function | Dirichlet beta function | Dirichlet eta function | Dirichlet lambda function | Dirichlet L-series | Dirichlet series | Epstein zeta function | Jacobi zeta function | nint zeta function | periodic zeta function | prime zeta function | Riemann zeta function | Selberg zeta function | zeta-regularized product

Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6

In[87]:= Zeta[2.0]
Out[87]= 1.64493

In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139

In[93]:= Table[1/k^2, {k, 1, 10}]
Out[93]= {1, 1/4, 1/9, 1/16, 1/25, 1/36, 1/49, 1/64, 1/81, 1/100}
```

Mathematica has a built in command, zeta, so we will look at Zeta[2] and Zeta[2] is a famous number $\pi^2/6$. You can evaluate this numerically and let me do that over here. Zeta of 2.0. The moment I put 2.0, Mathematica will interpret it as an approximate number and will calculate its numerical value. It is 1.64493. Now, as a series this is simply $1+1/2^2+1/3^2$ and so on.

That is the series we are talking about. Let us go ahead and calculate that and see if this gives me 1.64493, which is what zeta of 2.0 is. I see that this is away, so I may be tempted to, you know, add more terms to this to find out how quickly it converges to that, and we see that we are not quite there. And we can keep on doing this manually, add one time after another and compute it and we see that it is actually progressing very, very slowly.

As I add another term I see there is a small shift in pretty much you know the four digit or something, which is not, you know, impressive, I have to catch it to 1.64493. Of course, in order to get there, in order to get pi square by 6 exactly, I need to sum an infinite number of terms.

And that is something I am not going to do manually, of course, and even computationally, you cannot sum up to infinite number of terms, but you believe, at some point, if you sum up large number of terms, at some point, you will get a very accurate number, which is close to $\pi^2/6$. To six digits of accuracy it will be 1.64493.

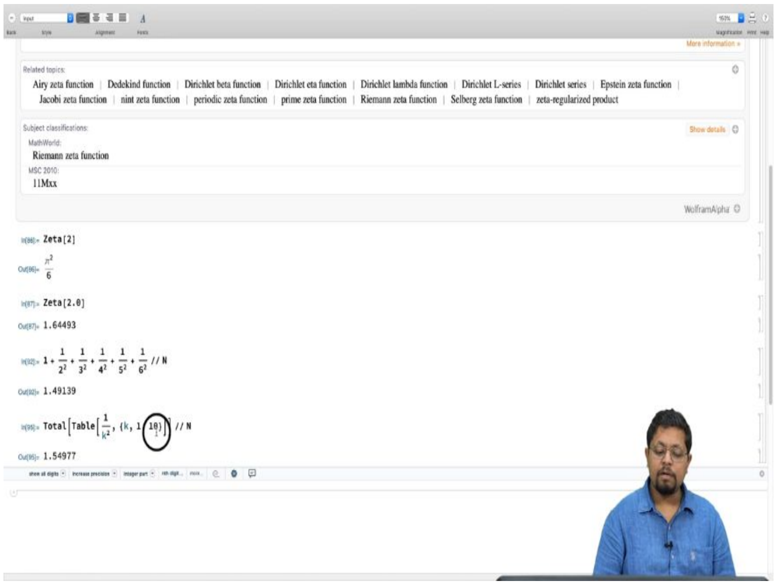
So, in order to get that number, let us go ahead and calculate this by adding a large number of terms. In order to add a large number of terms, I need to produce this sequence. In order to produce this sequence now, we can see I will do this many, many times, I have to do it a large number of times. So, I need to repeat something again and again and at this point, you would think that I should write a For loop. In order to write a For loop or some sort some sort of loop.

But rather than doing this in the For loop, first, what I will do is I will do this with a Table, which you are already familiar with. Table command is really interesting and powerful. So, let us do this with tables. So, in order to do that table first, what I will do is I will produce the sequence of these terms. And then I will sum them up.

So simple algorithm, I will produce a sequence to up to whatever desired numbers I want. And I will sum it up. Okay, so let us go ahead and produce this sequence. So, Table of $1/k^2$ is what I want to calculate Zeta[n] over here, n is 2, so $1/k^2$. And I want to sum k from 1 to, let us say 10. And that is my sequence of digits.

So, it is inverse of squares, integer squares. And if I want to find the sum of all these terms, all I need to do is total the terms in this list. So, the output of Table is a list and I want to add all the terms in the list.

(Refer Slide Time: 5:48)



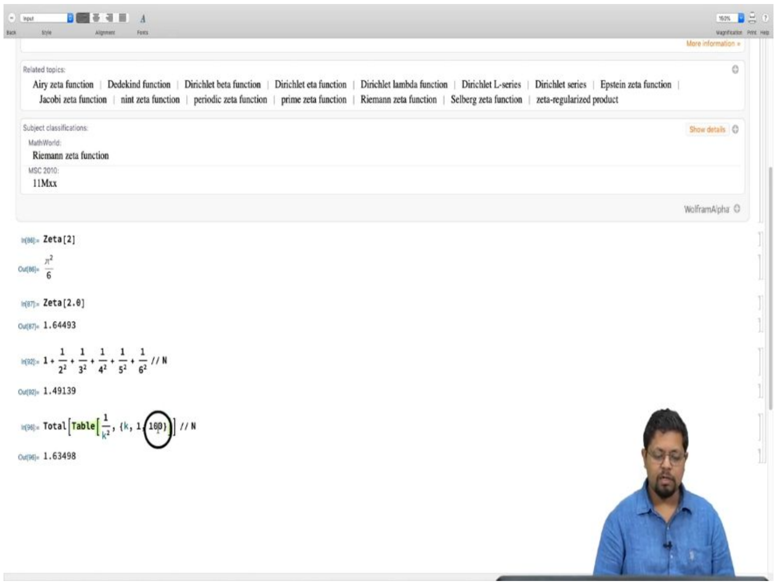
Related topics:
Airy zeta function | Dedekind function | Dirichlet beta function | Dirichlet eta function | Dirichlet lambda function | Dirichlet L-series | Dirichlet series | Epstein zeta function | Jacobi zeta function | nint zeta function | periodic zeta function | prime zeta function | Riemann zeta function | Selberg zeta function | zeta-regularized product

Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6

In[87]:= Zeta[2.0]
Out[87]= 1.64493

In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139

In[95]:= Total[Table[1/k^2, {k, 1, 10}]] // N
Out[95]= 1.54977
```

So I can go ahead and total them and that is what I get. Mathematica goes and sums them up, you know, in a rational number form, but I want to see its numerical value. So, I apply //N to it, and I will get 1.54977. So, that means 10 digits, 10 numbers are also not enough to reproduce this. So, let us go ahead and increase that, make it 100 this time and see what happens.

(Refer Slide Time: 6:12)



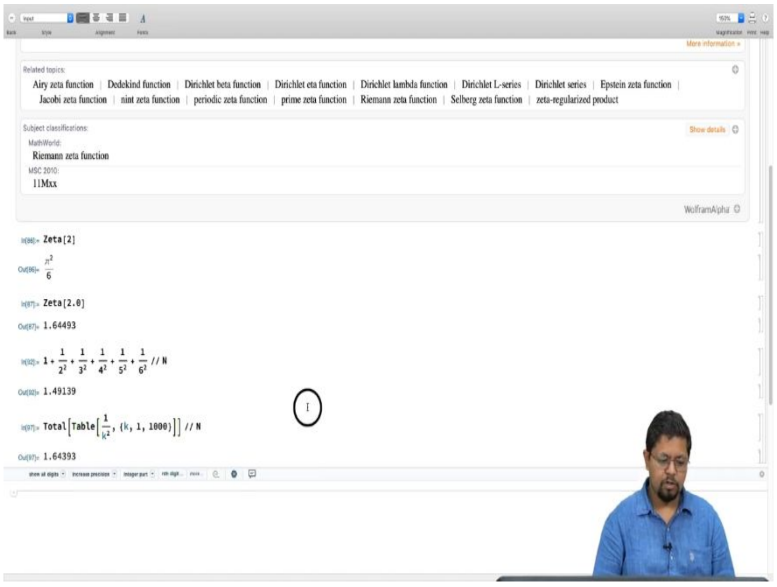
Related topics:
Airy zeta function | Dedekind function | Dirichlet beta function | Dirichlet eta function | Dirichlet lambda function | Dirichlet L-series | Dirichlet series | Epstein zeta function | Jacobi zeta function | nint zeta function | periodic zeta function | prime zeta function | Riemann zeta function | Selberg zeta function | zeta-regularized product

Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6

In[87]:= Zeta[2.0]
Out[87]= 1.64493

In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139

In[96]:= Total[Table[1/k^2, {k, 1, 100}]] // N
Out[96]= 1.63498
```

Okay, I am getting closer, let me make it 1000 maybe.

(Refer Slide Time: 6:18)



Related topics:
Airy zeta function | Dedekind function | Dirichlet beta function | Dirichlet eta function | Dirichlet lambda function | Dirichlet L-series | Dirichlet series | Epstein zeta function | Jacobi zeta function | nint zeta function | periodic zeta function | prime zeta function | Riemann zeta function | Selberg zeta function | zeta-regularized product

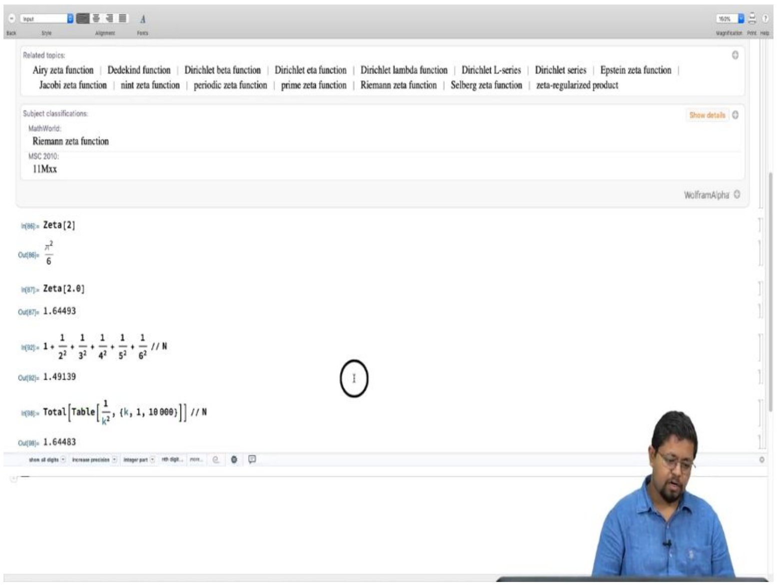
Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6
In[87]:= Zeta[2.0]
Out[87]= 1.64493
In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139
In[97]:= Total[Table[1/k^2, {k, 1, 1000}]] // N
Out[97]= 1.64393
```

Good, I can match three digits, but I am still not able to agree with three digits and as it happens with the case, it is 1/k^2 where k becomes progressively larger as I increase the number of terms in this series.

1/k^2 is suppressed and therefore this series progresses very, very slowly. It will take a long time or large number of terms to converge to this number 1.64493. And in this exercise, we are exactly going to do that and find out. Let us try 10,000 terms.

(Refer Slide Time: 06:46)



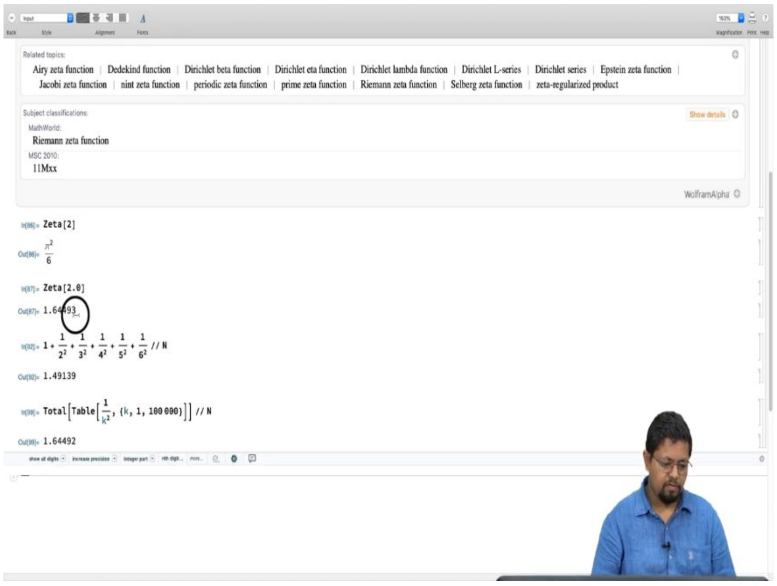
Related topics:
Airy zeta function | Dedekind function | Dirichlet beta function | Dirichlet eta function | Dirichlet lambda function | Dirichlet L-series | Dirichlet series | Epstein zeta function | Jacobi zeta function | nint zeta function | periodic zeta function | prime zeta function | Riemann zeta function | Selberg zeta function | zeta-regularized product

Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6
In[87]:= Zeta[2.0]
Out[87]= 1.64493
In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139
In[98]:= Total[Table[1/k^2, {k, 1, 10000}]] // N
Out[98]= 1.64483
```

Okay, we are getting closer but not quite there. First four digits agree in these two numbers but not quite the fifth and the six digits. So, in order to get that, let me go ahead and add one more 0 and see what happens.

(Refer Slide Time: 7:00)



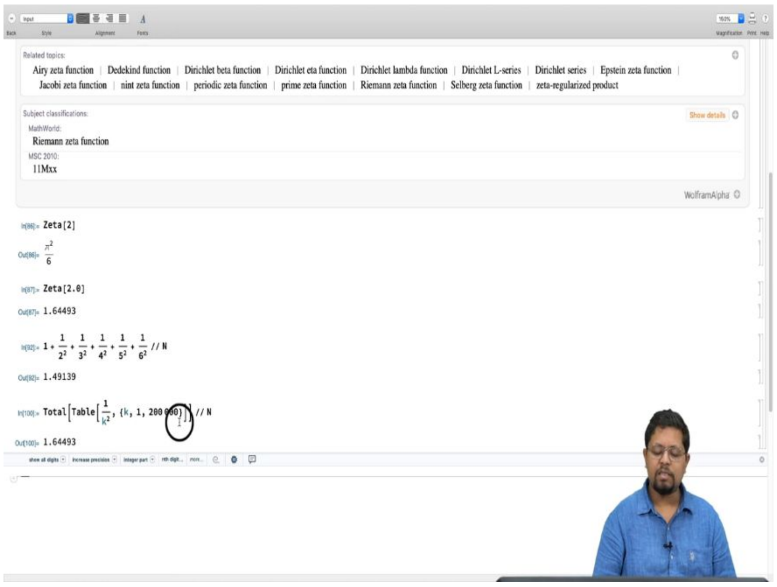
Related topics:
Airy zeta function | Dedekind function | Dirichlet beta function | Dirichlet eta function | Dirichlet lambda function | Dirichlet L-series | Dirichlet series | Epstein zeta function | Jacobi zeta function | nint zeta function | periodic zeta function | prime zeta function | Riemann zeta function | Selberg zeta function | zeta-regularized product

Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6
In[87]:= Zeta[2.0]
Out[87]= 1.64493
In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139
In[99]:= Total[Table[1/k^2, {k, 1, 100 000}]] // N
Out[99]= 1.64492
```

Oh, there we go. It is quite close, we are still away not agreeing on the six digit. So, let us go ahead and maybe put a 2 here.

(Refer Slide Time: 7:11)



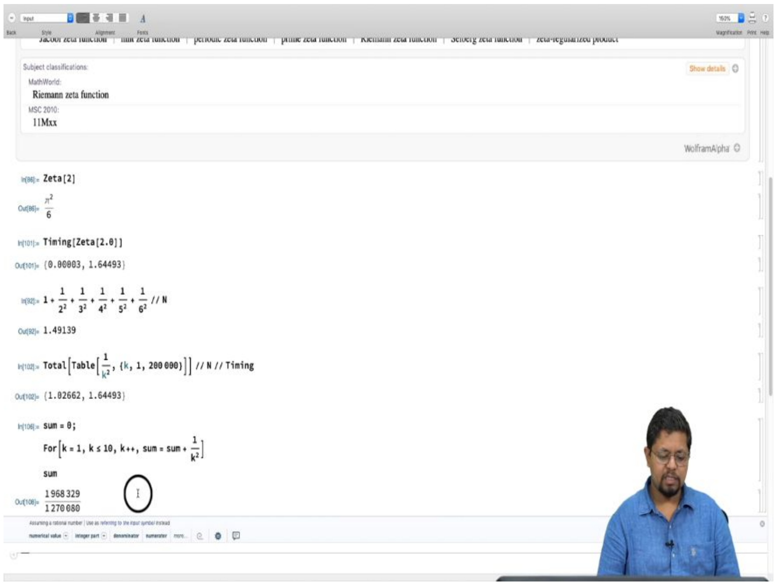
Related topics:
Airy zeta function | Dedekind function | Dirichlet beta function | Dirichlet eta function | Dirichlet lambda function | Dirichlet L-series | Dirichlet series | Epstein zeta function | Jacobi zeta function | nint zeta function | periodic zeta function | prime zeta function | Riemann zeta function | Selberg zeta function | zeta-regularized product

Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6
In[87]:= Zeta[2.0]
Out[87]= 1.64493
In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139
In[100]:= Total[Table[1/k^2, {k, 1, 200 000}]] // N
Out[100]= 1.64493
```

Wow, okay, there we go. So, at least I need something between a 100K and a 200K terms in order to get the zeta function accurately to six digits. Very well. So, it took us that many terms, let us find out how much time does it take to do this computation. In Mathematica, you can do that by actually putting all of this thing inside timing. So, doing slash slash is same thing as wrapping the whole thing inside Timing. So, let me show that over to you over here. So, I can do, I can go ahead and wrap this inside Timing.

(Refer Slide Time: 07:49)



Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
```

Out[86]= $\frac{\pi^2}{6}$

```
In[101]:= Timing[Zeta[2.0]]
Out[101]= {0.00003, 1.64493}
```

In[92]:= $1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\frac{1}{5^2}+\frac{1}{6^2}$ // N

```
Out[92]= 1.49139
```

In[102]:= Total[Table[$\frac{1}{k^2}$, {k, 1, 200 000}]] // N // Timing

```
Out[102]= {1.02662, 1.64493}
```

In[106]:= sum = 0;
For[k = 1, k ≤ 10, k++, sum = sum + $\frac{1}{k^2}$]
sum

Out[106]= $\frac{1968329}{1270080}$

When I do that I am going to get a list of two numbers. The first one is showing how much time did it take to compute this thing and the value of Zeta[2.0]. So, in order to compute Zeta[2.0] as a built in Mathematica function, it only took $3*10^{-5}$ seconds, this number is in seconds. So, it took like 3 to the power 10^{-5} seconds to compute this number using the built in function zeta.

But if I use my brute force series, or brute force sequence here, generate my brute force sequence to up to 200,000 terms and then add all those terms, turn them into a numerical value, all of this is going to take me how much time? To that I can add slash slash, which means whatever comes before this, apply the function timing on that.

Applying the function timing on that this is what I get. You see, it takes some time and it takes about, on this particular computer, it takes me about 1.02 seconds to compute this number. So, running this 200,000 iterations, it took me about 1 second to generate this number 1.64493.

Let us go ahead and do this in the For loop. Of course in Table, we generated a sequence but in For loop, I can iteratively add terms to whatever number of terms I want. So, in order to write a For loop, I want to define my sum equal to 0, I want to initialize my sum equal to 0 and then I want to write my For loop. And I am going to write the For loop here and also explain you the syntax at the same time.

So, a For loop has four arguments separated by commas. So, let me put three commas. Each of these spaces will take in an argument with the For function. The first argument of the For function is initialization. In order to initialize something over here, I am going to say n=0. I am going to start with n=0, that is my initialization, in the second argument of the For loop, I am going to specify what is the condition on n.

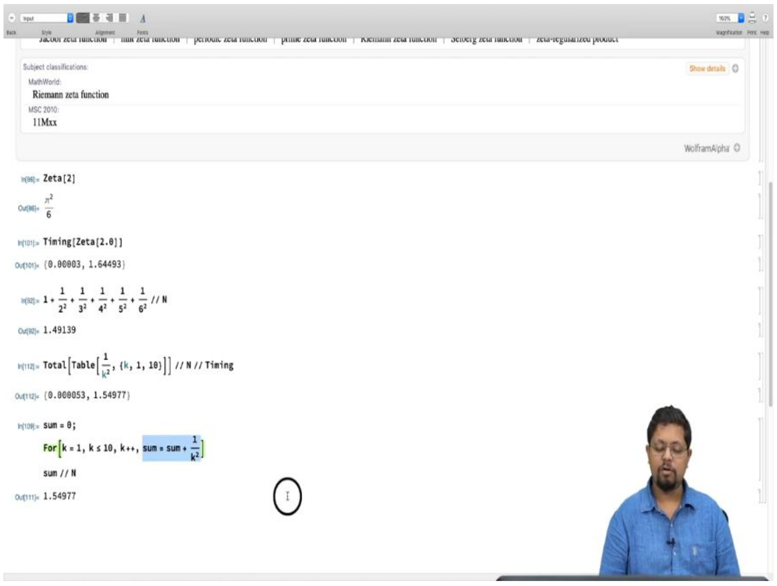
So n, I want to run from n equal to less than equal to, let us say, 10. We will make it 200,000 in a moment, but let me just limit this to 10. It is always a good idea to test how do you loop for small number of terms. Once you know that it is working fine, it is giving results that you expect, then you can go ahead and run it for a longer duration.

So, I am just going to constraint $n \leq 10$ over here. So, n=0, $n \leq 10$, that is my condition. I am going to say increment n that means after each round of running the body, executing the body, increment n by 1. So, n plus plus means the same thing as n = n+1.

So, n = n +1 means increment n by one. And n++ is a shorthand for that. It is also there in many other languages, such as C and C++. And I am sure many other languages use this notation. So, we will just use n plus plus and now the fourth argument, which is the most important argument of the For function is the body of the For function. What are you going to do in each iteration? So, in the for function, in each iteration, I am going to add to sum equal to sums in previous value plus $1/n^2$.

In fact, I should have called this as k, it is more appropriate to call this as k because that is what we are taking over here k++. And I will say $1/k^2$. That is my For loop. And once I am done with this For loop, I want to print sum and let me suppress any output for For loop. I should start from k=1, it says 1/0 infinity encountered. k does not start from 0, k starts from one. Let me make the correction and execute this again, I get this number over here.

(Refer Slide Time: 11:42)

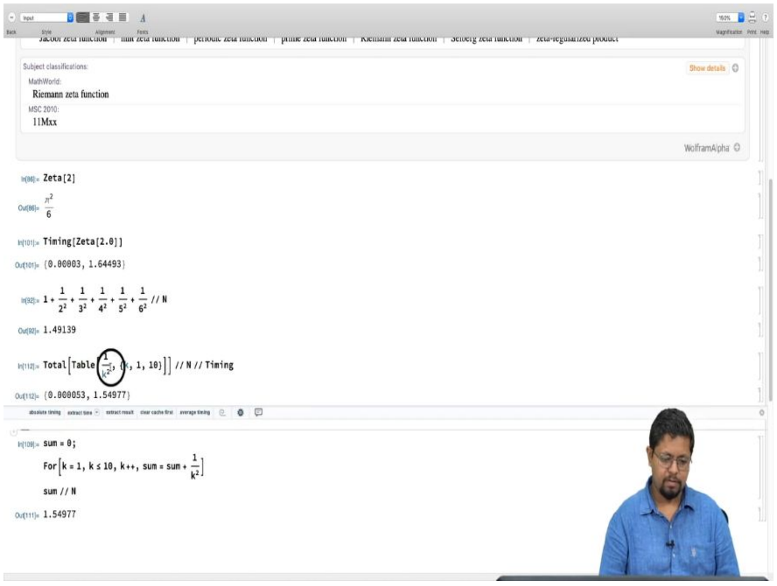


Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6

In[101]:= Timing[Zeta[2.0]]
Out[101]= {0.00003, 1.64493}

In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139

In[112]:= Total[Table[1/k^2, {k, 1, 10}]] // N // Timing
Out[112]= {0.000053, 1.54977}

In[109]:= sum = 0;
          For[k = 1, k ≤ 10, k++, sum = sum + 1/k^2]
          sum // N
Out[111]= 1.54977
```

Let me say, slash slash N again, sum so it is evaluated into numerical value, I get 1.54977, it is exactly the value that I get over here when I had this number up to 10. Okay?

(Refer Slide Time: 11:53)

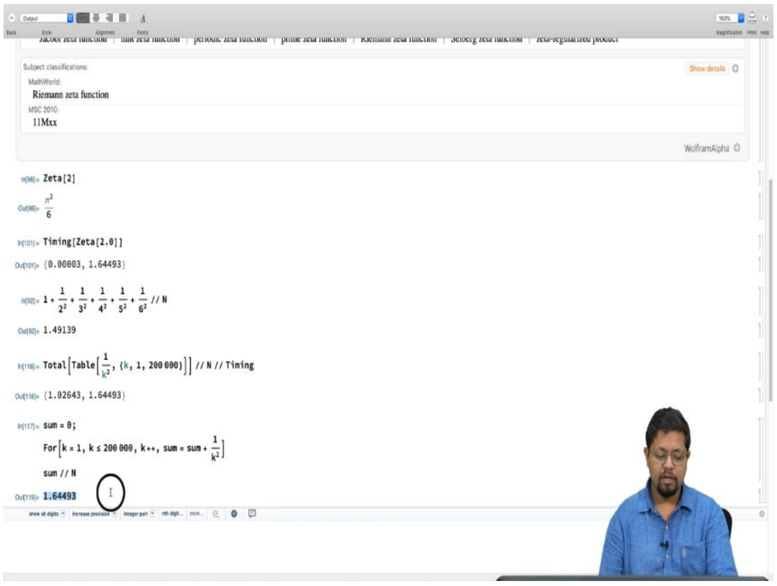


Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6

In[101]:= Timing[Zeta[2.0]]
Out[101]= {0.00003, 1.64493}

In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139

In[112]:= Total[Table[1/k^2, {k, 1, 10}]] // N // Timing
Out[112]= {0.000053, 1.54977}

In[109]:= sum = 0;
          For[k = 1, k ≤ 10, k++, sum = sum + 1/k^2]
          sum // N
Out[111]= 1.54977
```

So, that is my little code. Now all I have to do for the For loop so For loop is very simple that way. There are four arguments in the For loop initialization, condition, as long as this condition holds true, For loop will continue to execute the body, if this condition is not true, it will stop. k++ increments k and this is the body. So, initialization, condition, increment and fourth argument is the body. So, body is sum equals sum+1/k^2. That is my For loop. Now, I want to run it for a large number of terms.

(Refer Slide Time: 12:33)



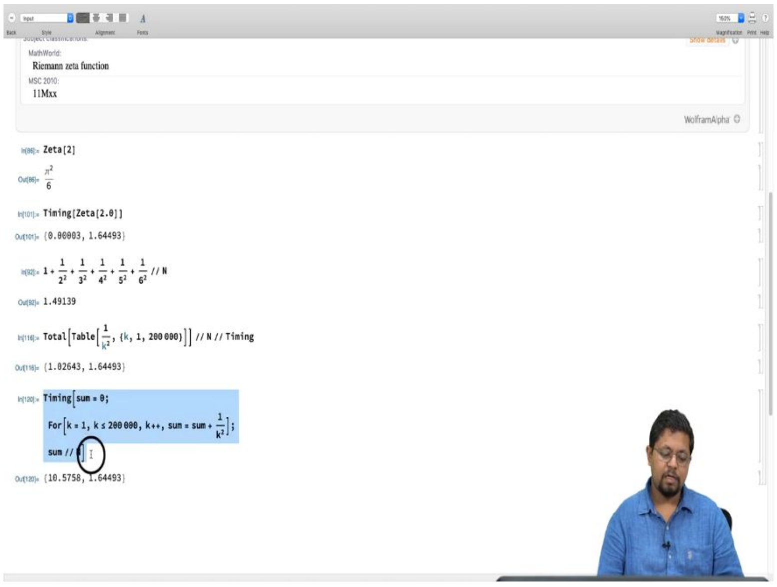
Subject classifications:
MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6

In[101]:= Timing[Zeta[2.0]]
Out[101]= {0.00003, 1.64493}

In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139

In[116]:= Total[Table[1/k^2, {k, 1, 200 000}]] // N // Timing
Out[116]= {1.02643, 1.64493}

In[117]:= sum = 0;
          For[k = 1, k ≤ 200 000, k++, sum = sum + 1/k^2]
          sum // N
Out[119]= 1.64493
```

In particular, I want to do this for 200,000. So, 200,000 and over here, let me do this again for the Table command. And you see that it takes me about 1 second. And I am going to do the same thing over here for k less than 200,000 and let me execute this. And you see it is going to take some time, it is already more than a couple of seconds, it is still running.

It is going to take some time and that is what I want to show it to you that For loop is not always a very efficient way of doing things. So, yeah, there we go, this execution finished, we got the number that we expected 1.64493 for the Riemann zeta function evaluated at two. Let us go ahead and check exact timing for this thing.

(Refer Slide Time: 13:27)



MathWorld:
Riemann zeta function
MSC 2010:
11Mxx

```
In[86]:= Zeta[2]
Out[86]= π^2/6

In[101]:= Timing[Zeta[2.0]]
Out[101]= {0.00003, 1.64493}

In[92]:= 1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + 1/6^2 // N
Out[92]= 1.49139

In[116]:= Total[Table[1/k^2, {k, 1, 200 000}]] // N // Timing
Out[116]= {1.02643, 1.64493}

In[120]:= Timing[sum = 0;
           For[k = 1, k ≤ 200 000, k++, sum = sum + 1/k^2];
           sum // N]
Out[120]= {10.5758, 1.64493}
```

So to do that, I will wrap this whole thing inside this timing command and press semicolon over here, I will run this thing. This will measure me exactly how much computational time on this particular computer on this particular system it took me to execute this value. From system to system, computer to computer it can change. In this case, I get the timing over here, it is 10.57 seconds and this is my value for the execution. So, in order to obtain Riemann zeta function zeta of 2.0 very accurately up to six digits.

I need to run up to, I need to consider 200,000 terms. And in order to do that, if I do it by For loop, which is what many of you who are familiar with programming, you are used to writing For loops like that. If you do write a For loop like that, you see it is not a very efficient way of doing things. And it takes about 10 seconds to compute the same thing what you could do it over here in 1 second, in about 1 second.

Of course, built in functions that are there in Mathematica, they use some advanced algorithms, advanced techniques to compute those functions. And these series expansions may not be very efficient. But for other cases where you know the series expansion, series expansion what you are using, the table command will often turn out to be more suitable than the For loop itself.

Now, so does that mean that we should never use really use For loop? My general advice is do not use For loop until and unless it is extremely necessary. So, when is it For loop that is extremely necessary? Whenever we can write Table command or some other way of generating the term without invoking For loop, we should always avoid it.

For loops are not very efficient. And we should, when should we use For loop? We should use For loops when there is no other way left. That means, my decision for this iteration depends on what happened before, what happened in the previous iteration. When it depends on that, that is when I should use For loop, that is my execution of this step is dependent on the history. When my execution of this step depends on the history that is the body depends on the history, that is when I should use a For loop.

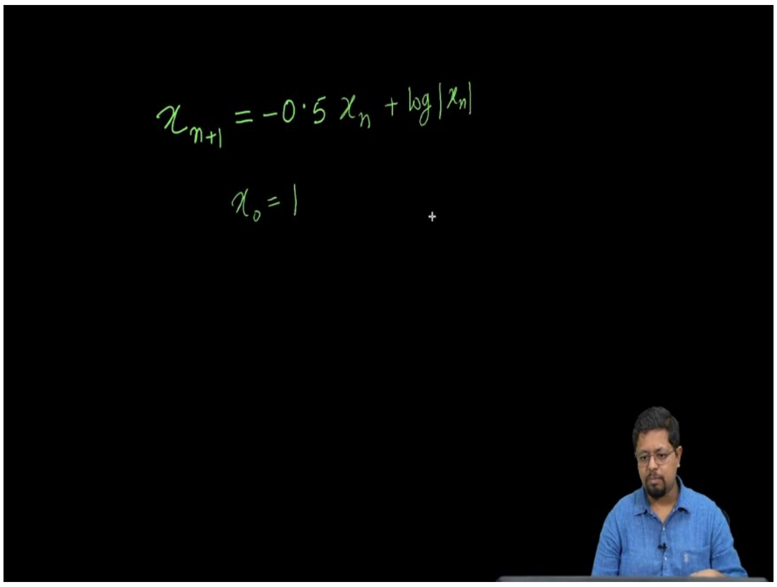
Now over here, it appears when I am executing this command, that I need to know the previous sum in order to execute the sum at this moment. I am adding the increment to the sum over k square. In order to add that I need to know what is the sum in the previous moment, but that is an illusion, because I just need to know each term in the series and if I know each time in the series all I can do is simply go ahead and sum it up. Because that is what we did over here and we found that that was much, much more efficient. Why?

The internal reason is that table commands tries to generate a sequence. So, it is acting directly on the in the memory on on a list or an array. So, it is very efficient that way it is the processor is working at the level, the memory, it is trying to do the same things again and again. So, in this case it was generating a sequence of 1 over k square again and again and it generated a memory, array in the memory and that was very, very efficient.

While in the For loop, every time we are executing the body, we are going back reading, updating k, reading a k value from the memory, incrementing the body, then going back again, and doing this repeatedly again and again. And in this case, it was actually very simple to just simply generate that entire sequence in one go.

So, that is why Table command ends up being more efficiently. Now, let us take an example where table command is not going to work and you really need to know, you really need to find out, you really need to use the for loop. So, here is an example.

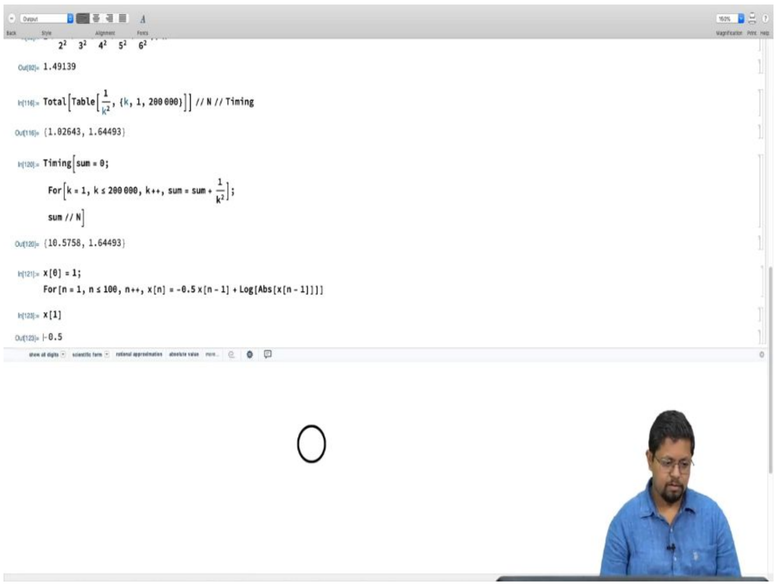
(Refer Slide Time: 17:05)



Suppose I need to solve this difference equation $x_{n+1} = -0.5\, x_n + \log|x_n|$. Let us just take this difference equation. Now, this difference equation, in order to find x_{n+1}, I need to know what is x_n. And in order to make this complete, let us say $x_0 = 1$. So, we start from $x_0 = 1$ and we have to find x_1, x_2, x_3 and so on.

But every time I want to find the next value of x_n, I need to know what is the previous value of x_n. In order to solve this equation, I need to go and write a For loop because this is something I cannot do with the Table command. I cannot generate the entire sequence, I do not know the entire sequence in one go, I have to depend on the result of the previous step in order to find out what is happening now. So, let us go ahead and do this by writing a For loop. So, here is my For loop.

(Refer Slide Time: 18:10)

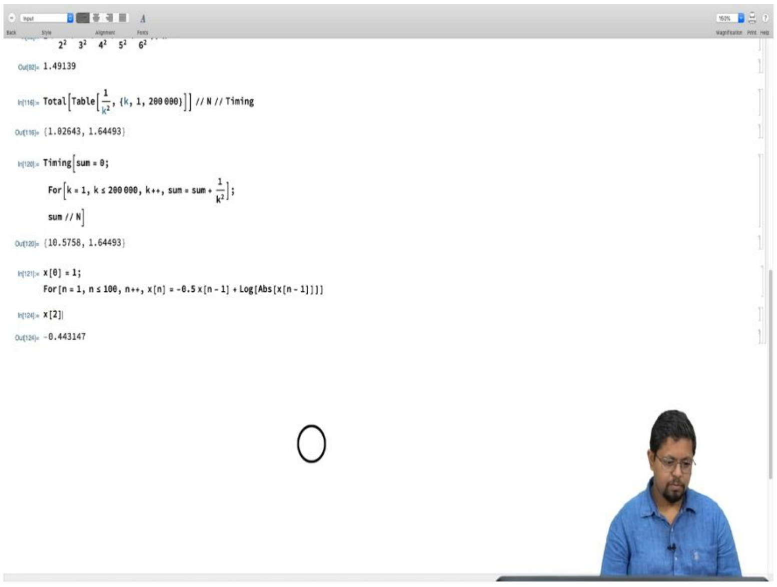


```
        2²   3²   4²   5²   6²

Out[92]= 1.49139

In[116]:= Total[Table[1/k², {k, 1, 200 000}]] // N // Timing

Out[116]= {1.02643, 1.64493}

In[120]:= Timing[sum = 0;
           For[k = 1, k ≤ 200 000, k++, sum = sum + 1/k²];
           sum // N]

Out[120]= {10.5758, 1.64493}

In[121]:= x[0] = 1;
          For[n = 1, n ≤ 100, n++, x[n] = -0.5 x[n - 1] + Log[Abs[x[n - 1]]]]

In[123]:= x[1]

Out[123]= -0.5
```

I will go ahead and say, in order to generate the entire sequence, what I will do is I will say, $x_0 = 1$ that is my initialization. And I will say, n runs from 1 to n ≤ 100. Let us say if we want to do this for 100 iterations, increment n by 1 each time. And coming to the body.

In the body, I want to find out what is x_n and that should be equal to $-0.5\, x_0 + \log|x|$, which is absolute value of x_{n-1}. This should also have been n-1, not 0. All right, so let me go ahead and execute this. This looks like a very nested complicated structure. Let us go ahead and execute this For loop. And this is going to, the execution of this is done, you can see that if I type x_1, x_2, x_3.

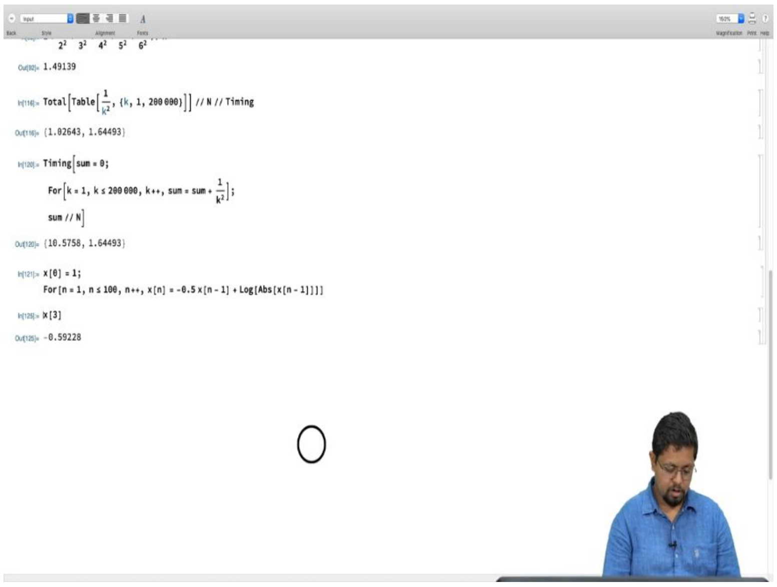
(Refer Slide Time: 19:27)



```
        2²   3²   4²   5²   6²
Out[92]= 1.49139

In[116]:= Total[Table[1/k², {k, 1, 200 000}]] // N // Timing

Out[116]= {1.02643, 1.64493}

In[120]:= Timing[sum = 0;
           For[k = 1, k ≤ 200 000, k++, sum = sum + 1/k²];
           sum // N]

Out[120]= {10.5758, 1.64493}

In[121]:= x[0] = 1;
          For[n = 1, n ≤ 100, n++, x[n] = -0.5 x[n - 1] + Log[Abs[x[n - 1]]]]

In[124]:= x[2]

Out[124]= -0.443147
```

I have got those numbers.

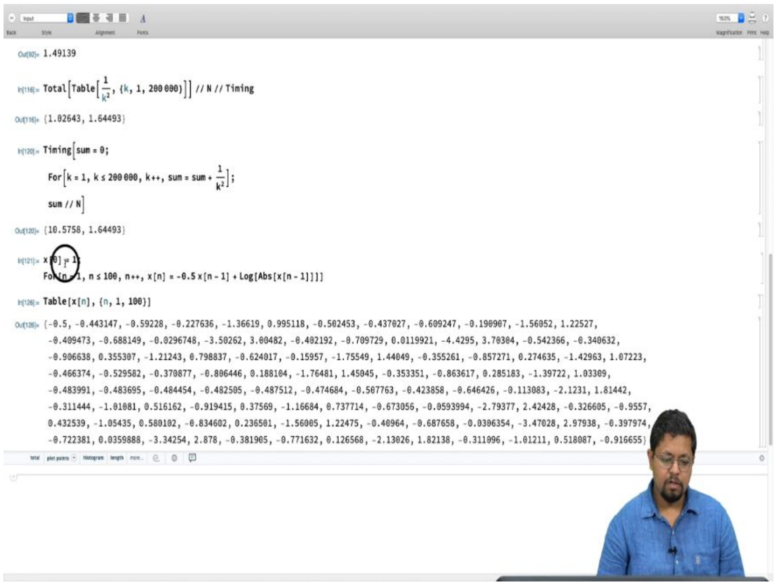
(Refer Slide Time: 19:30)



```
        2²   3²   4²   5²   6²
Out[92]= 1.49139

In[116]:= Total[Table[1/k², {k, 1, 200 000}]] // N // Timing

Out[116]= {1.02643, 1.64493}

In[120]:= Timing[sum = 0;
           For[k = 1, k ≤ 200 000, k++, sum = sum + 1/k²];
           sum // N]

Out[120]= {10.5758, 1.64493}

In[121]:= x[0] = 1;
          For[n = 1, n ≤ 100, n++, x[n] = -0.5 x[n - 1] + Log[Abs[x[n - 1]]]]

In[125]:= x[3]

Out[125]= -0.59228
```

In order to print the entire series, I will simply say table xn and n goes from 1 to 100.

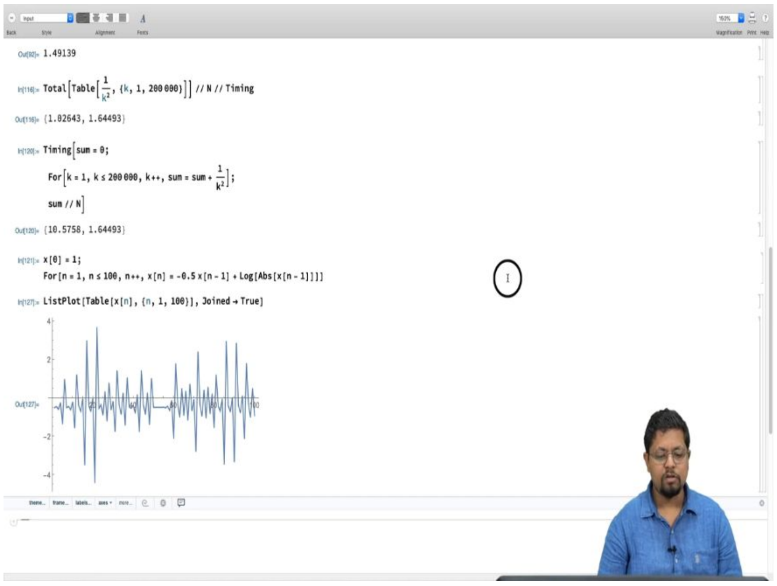
(Refer Slide Time: 19:32)



```
Out[92]= 1.49139

In[116]:= Total[Table[1/k^2, {k, 1, 200000}]] // N // Timing

Out[116]= {1.02643, 1.64493}

In[120]:= Timing[sum = 0;
            For[k = 1, k ≤ 200000, k++, sum = sum + 1/k^2];
            sum // N]

Out[120]= {10.5758, 1.64493}

In[121]:= x[0] = 1;
          For[n = 1, n ≤ 100, n++, x[n] = -0.5 x[n - 1] + Log[Abs[x[n - 1]]]]

In[126]:= Table[x[n], {n, 1, 100}]

Out[126]= {-0.5, -0.443147, -0.59228, -0.227636, -1.36619, 0.995118, -0.502453, -0.437027, -0.609247, -0.190907, -1.56052, 1.22527,
  -0.409473, -0.688149, -0.0296748, -3.50262, 3.00482, -0.402192, -0.709729, 0.0119921, -4.4295, 3.70304, -0.542366, -0.340632,
  -0.906638, 0.355307, -1.21243, 0.798837, -0.624017, -0.15957, -1.75549, 1.44049, -0.355261, -0.857271, 0.274635, -1.42963, 1.07223,
  -0.466374, -0.529582, -0.370877, -0.806446, 0.188104, -1.76481, 1.45045, -0.353351, -0.863617, 0.285183, -1.39722, 1.03309,
  -0.483991, -0.483695, -0.484454, -0.482505, -0.487512, -0.474684, -0.507763, -0.423858, -0.646426, -0.113083, -2.1231, 1.81442,
  -0.311444, -1.01081, 0.516162, -0.919415, 0.37569, -1.16684, 0.737714, -0.673056, -0.0593994, -2.79377, 2.42428, -0.326605, -0.9557,
  0.432539, -1.05435, 0.580102, -0.834602, 0.236501, -1.56005, 1.22475, -0.40964, -0.687658, -0.0306354, -3.47028, 2.97938, -0.397974,
  -0.722381, 0.0359888, -3.34254, 2.878, -0.381905, -0.771632, 0.126568, -2.13026, 1.82138, -0.311096, -1.01211, 0.518087, -0.916655}
```

And there is my entire series. I could not have predicted these terms by writing a simple table command, I needed to actually write a For loop. If you want, you can go ahead and apply a list plot on this just to see how these numbers varied with each iteration. So, I will say joined as true.

(Refer Slide Time: 19:53)

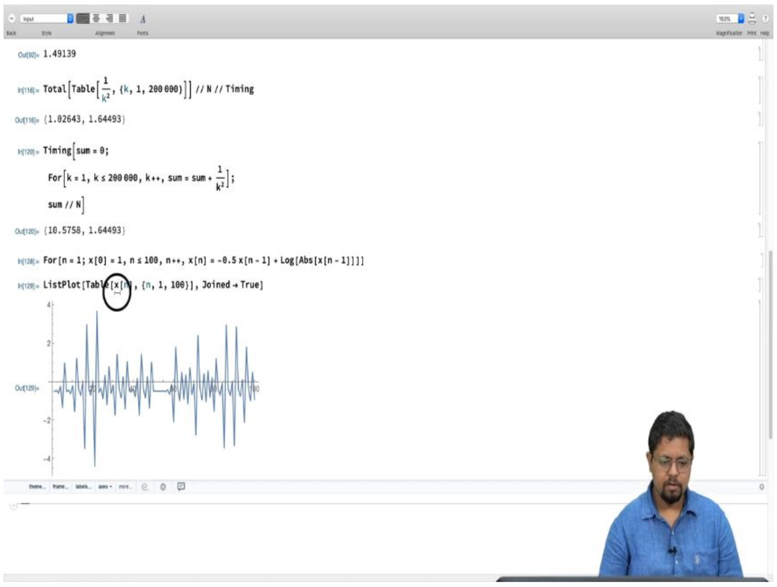


```
Out[92]= 1.49139

In[116]:= Total[Table[1/k^2, {k, 1, 200000}]] // N // Timing

Out[116]= {1.02643, 1.64493}

In[120]:= Timing[sum = 0;
            For[k = 1, k ≤ 200000, k++, sum = sum + 1/k^2];
            sum // N]

Out[120]= {10.5758, 1.64493}

In[121]:= x[0] = 1;
          For[n = 1, n ≤ 100, n++, x[n] = -0.5 x[n - 1] + Log[Abs[x[n - 1]]]]

In[127]:= ListPlot[Table[x[n], {n, 1, 100}], Joined → True]

Out[127]=
```

You see, it is some sort of oscillation. So, this difference equation leads to some sort of oscillation. In this particular case, I could have not imagined how to do this from a Table command, so I had to resort to a For loop.

Once again, I can make this For loop little bit more compact, sometimes it is useful, but that can also make reading a little bit more difficult. So, for example, I can go ahead and I could in the initialization itself, I could have said n = 1 and I could have added another statement in the initialization part by separating them by semicolons.

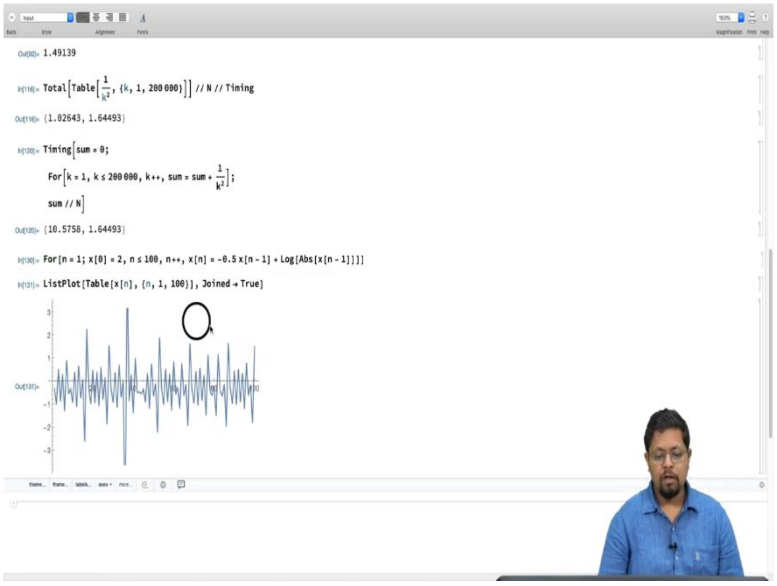
(Refer Slide Time: 20:33)



```
Out[92]= 1.49139

In[116]:= Total[Table[1/k^2, {k, 1, 200 000}]] // N // Timing

Out[116]= {1.02643, 1.64493}

In[120]:= Timing[sum = 0;
           For[k = 1, k ≤ 200 000, k++, sum = sum + 1/k^2];
           sum // N]

Out[120]= {10.5758, 1.64493}

In[128]:= For[n = 1; x[0] = 1, n ≤ 100, n++, x[n] = -0.5 x[n - 1] + Log[Abs[x[n - 1]]]]

In[129]:= ListPlot[Table[x[n], {n, 1, 100}], Joined → True]

Out[129]=
```

I could have said $x_0 = 1$. Let us go ahead and do this. Let us try this with $x_0 = 2$ and you can go ahead and play around with it.

(Refer Slide Time: 20:49)



```
Out[92]= 1.49139

In[116]:= Total[Table[1/k^2, {k, 1, 200 000}]] // N // Timing

Out[116]= {1.02643, 1.64493}

In[120]:= Timing[sum = 0;
           For[k = 1, k ≤ 200 000, k++, sum = sum + 1/k^2];
           sum // N]

Out[120]= {10.5758, 1.64493}

In[130]:= For[n = 1; x[0] = 2, n ≤ 100, n++, x[n] = -0.5 x[n - 1] + Log[Abs[x[n - 1]]]]

In[131]:= ListPlot[Table[x[n], {n, 1, 100}], Joined → True]

Out[131]=
```

The important point to note here is that each of these arguments of the for, the initialization part can contain multiple statements separated by semicolons. Again the condition can contain, a condition should can contain multiple statements as well, but the last statement should evaluate to a true or a false value. If it is true, it will continue to execute the loop that is execute the body and continue to increment.

The increment does not always have to be n++ it can be anything that you like as long as it tells you that this is what you mean by going to the next step. So, increment can be anything, incrementation can contain multiple different steps. In fact, if I wanted I can include this body also inside the increment and left the body as blank.

And the body is of course the main part of the for loop where you are executing your main command. As we go ahead and progress in this course, we will use this For loop for writing code for Runge-Kutta algorithm, Euler’s method for solving differential equations and so on. So, this was a brief introduction to For loop, its efficiency and its comparison with the Table command. We will see you next time.