IIT Madras

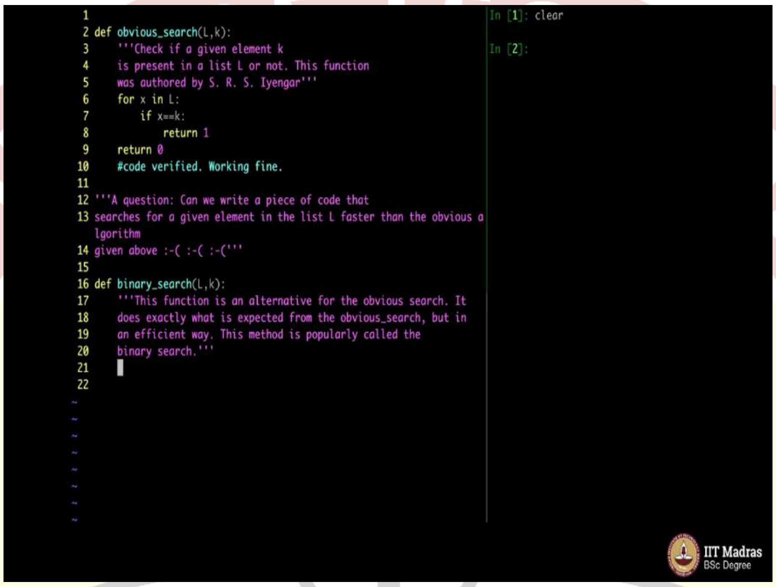
ONLINE DEGREE

Programming in Python

Professor Sudarshan Iyengar
Department of Computer Science and Engineering
Indian Institute of Technology, Ropar
Mr. Omkar Joshi
Course Instructor
Indian Institute of Technology, Madras
Online Degree Programme

Binary Speech Implementation

(Refer Slide Time: 00:16)



```
def obvious_search(L,k):
    '''Check if a given element k
    is present in a list L or not. This function
    was authored by S. R. S. Iyengar'''
    for x in L:
        if x==k:
            return 1
    return 0
    #code verified. Working fine.

'''A question: Can we write a piece of code that
searches for a given element in the list L faster than the obvious a
lgorithm
given above :-( :-( :-('''

def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''
```

```
In [1]: clear

In [2]:
```

So, let us start with our magic function that will perform much better than obvious search. Let us see what that is going to be. So, I want to call it binary search as discussed, it will take a list L and search for the element k, let me write that as a comment here. This function is a alternative for the obvious search. It does exactly what is expected from the obvious search, but in an efficient way. What is efficient here?

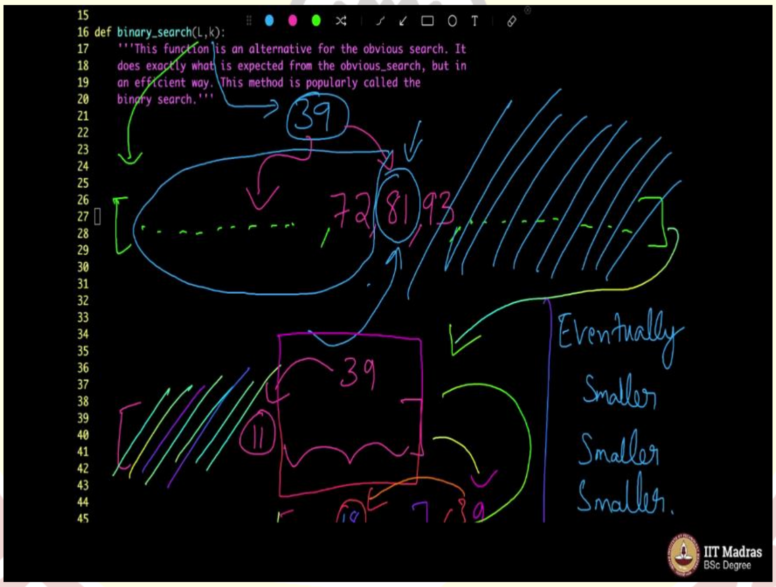
This method is popularly called the binary search. This is one thing that you want to really learn in this entire subject called Python, let it be the binary search, it is just in magical. You will see the magic happen.

Alright. So, how do we go about it? I am going to take, I am going to patiently write the code here, I want to send out a small warning message here, it might appear like, and once again, I am struggling, it might appear like I am writing a longish code, instead of a short and sweet and a crisp code. I am purposely going to write a long code here. And many people out there who want to sort of show that they can write a smaller code be it as it may, the point is not to write a small code here, the point is to write a piece of code that you all can understand.

So, please bear with me, my code can be a little longer. You can always Google up and then see a code that is short, sweet and finishes in just half a page, but then the motive is not there. The motive is to think on the fly. I am going to patiently connect what we were discussing before. On the dictionary idea, the Hema Malini example onto search.

So, as and always, there will be some geek, who will look up geeks for geeks code and say, there is a much easier way to do it. The point is not that, the point is to make you all understand, so please bear with me. Small is not always better, write I mean, sometimes you may want to write longish code, so that you explain things well. A two-and-a-half hours Bollywood movie can be summarized in two minutes, but then it is fun to watch it for two-and-a-half hours wrong analogy, bad example, but what it is the spirit with which I am saying. So, without any further cacophony, let me start off with the code.

(Refer Slide Time: 03:21)



```
def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''
```

So, what do we do now? Let me think. I will ensure that this crawls a political just so that you can see it. So, what I will do is I will try to zoom this, perfect. Let me just explain a few things here. So, there is an array, right, you remember. This is an array, and in this very array you will have many elements, this is your list, basically, L. The list L, you have some, you have a list L, whatever is here L is this, and it is sorted this is sorted.

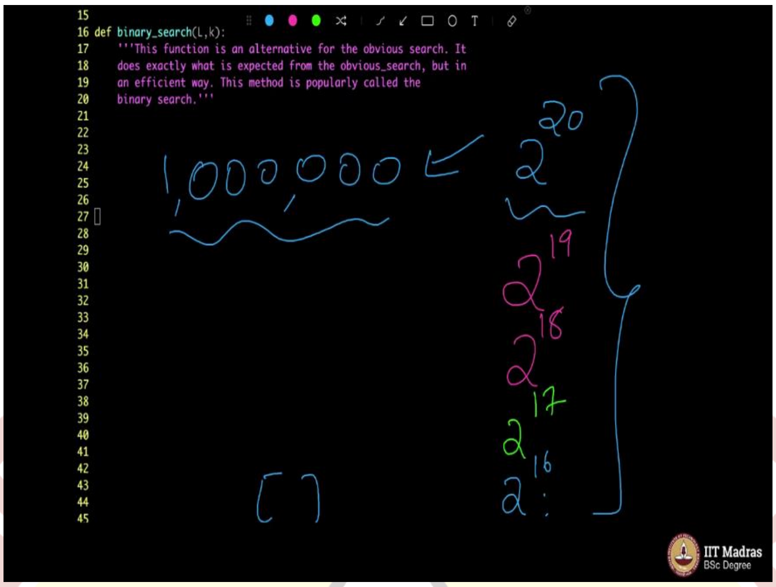
The previous obvious search, it works on any list, but what we are going to write right now, only works on sorted list. So, throughout the discussion, we are going to assume that the list L is sorted. You are going to remember this. Alright, so let me remove that for the time being now that you know that the list is sorted. So just in case you had numbers, let us say here 72, 81, 93, and so on. If your k where to be let us say some 39, how will you find if 39 is present in this list or not.

In this very list, you need to find whether 31 is present or not, how will you find that. I will first come to the midpoint. The idea is very simple. I will come to the midpoint, midpoint here let us say was 81, and I will check 39 is definitely less than 81. So, what I will do is, I will now realize that this particular thing, 39 should be on the left side of this. And I can simply chop off this part of the list.

I do not need this part of the list, because I need to just find if 39 is present on the left side. So, the point is 39, you take the left side of the list, and then check if 39 is present here or not. How do you do that? Again, what you do is look at the middle element here, middle or any element will do, middle element seems to be a wise idea, you see. I mean, you are cutting it by half, example if you remember. So, middle element happens to be 11 here, then your 39 will be on this side.

It will be on this side, 39 will be on the right side. So, what you do is you chop off this side of the list, and concentrate on the presence of 39 on the right side of the list. And then what you do is you again, look at this part of the list and then look at the center point, and then see if 39 is present there or not. Just in case you see 18 here, you know very well 39 should be this place. You see, you are making the list goes half in size and then half in size. And then eventually, the list becomes smaller and smaller and smaller. So small. When you take a number and keep halving it, you keep making that go half and half and half, you know what happens.

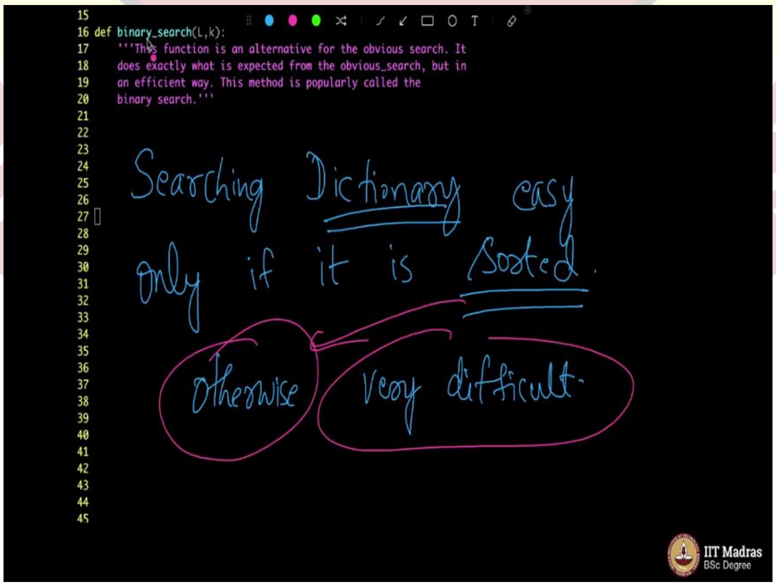
(Refer Slide Time: 07:17)



```python
def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''
```

So, remember, so you take a very, very big number, let us say, 1 million, half of this. 1 million is roughly 2 to the power of 20. In fact, 2 to the power of 20 is 1024 into 1024, so approximately I will call this 2 to the power of 20, and then when you half this, when you make half of this you get 2 to the 19, not 2 to the 10. So, and then half of this will be 2 to the 18, and half of this will be 2 to the 17, and half of this will be 2 to the 16, and so on. Roughly in 20 steps. In fact, indeed, in 20 steps, you will make a list of size 1 million become a list of size, just one. Think about what I am saying.

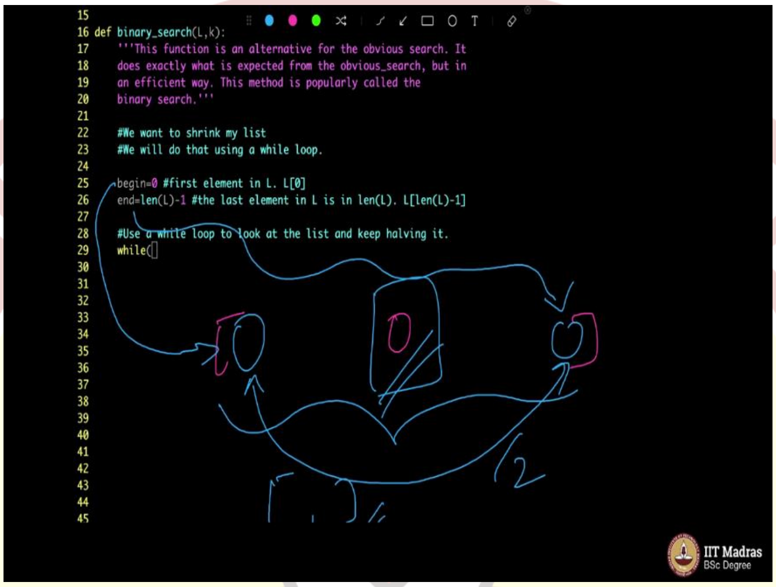
(Refer Slide Time: 08:09)



```python
def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''
```

I mean, it is a very powerful idea, but the entire thing will work you can search through a list only if the list is sorted. You can search a dictionary. Let me make a note of this, this is an important point. So, you can, searching a dictionary is easy. Searching a dictionary is easy. By easy I mean, super easy only if it is sorted. By sorted I mean, it is arranged in alphabetical order. Otherwise, it is not easy. Otherwise, it is going to be very difficult. And not just difficult I mean, it is close to impossible to search where a word is in a dictionary, if it is not sorted. If it is not sorted, otherwise means not sorted it is going to be very difficult.

(Refer Slide Time: 09:26)



```
def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while()
```

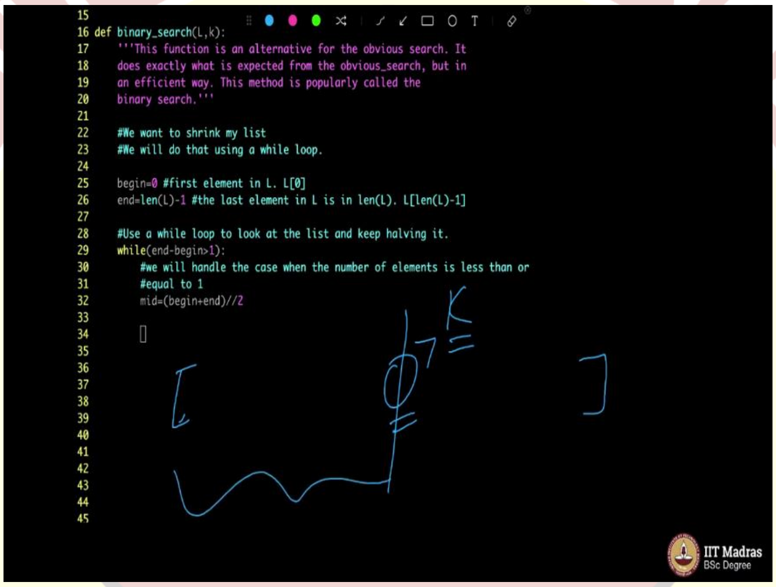
So, I am going to assume that the list L is sorted. And then I am going to find the element k in the list L. So let us go, get going, and then write a piece of code. Let me slowly think, how will my code look like. What did I do? I want to shrink my vision in the list. Basically, the list should become smaller and smaller. Remember what I illustrated? I want to shrink my list.

How do I do that? I will do that using a while loop it is always a good idea to say. We will do that using, when you write a research paper, they say we should always say we. It means the reader as well as the researcher both of us put together, we saw that, we did that, we will take a look at this plot that is the lesson we will learn in writing research papers. Anyways, we will say we will do that using a while loop, so what exactly will we do? We will say while firstly, the first element in the list will be 0, zeroth element of L.

Basically, this is first element in L, index of the first element in L, not necessarily, basically, the first element in L will be L of 0, and the last element in L will be basically a len of L minus 1. The last element in L is in len of L. Basically, I am talking about L of len of L minus 1, I am talking about this. Here, I am talking about L of 0. I think these comments are just complicating things, but then you will get to know what I am trying to do as we proceed.

Use a while loop to stay, to look at the list and keep halving it, halving it is the word. So, what do I do? I will say, while you see, let me just illustrate that. You see there is a list. I need to find the middle of this list. How do I find the middle of this list? I should take the beginning, the last, and then this plus this by 2 will give me this. So, this is basically your begin. This is basically your end, and the midpoint will actually be, begin plus end by 2. I should keep doing this, as long as this list is big enough. What do you mean by big enough? By big enough, I mean, it has at least some, let us say two elements, otherwise, it does not make sense. See at least two elements.

(Refer Slide Time: 12:10)



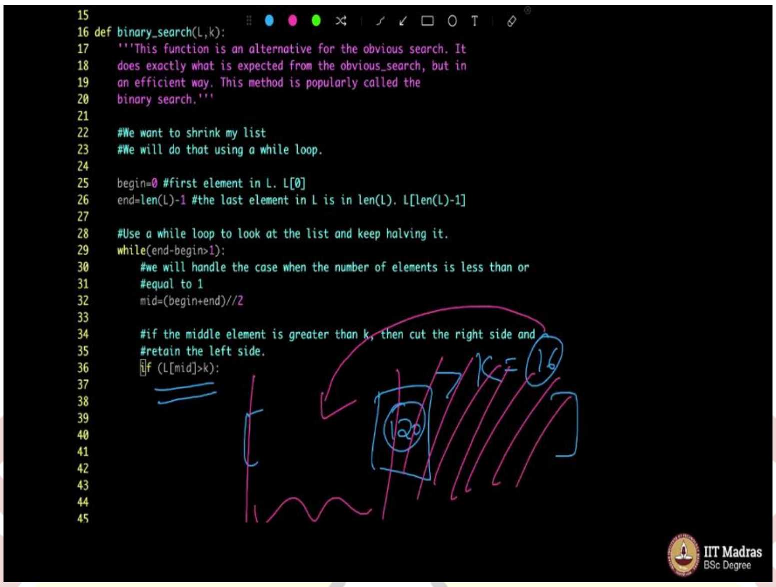
```
def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1
        mid=(begin+end)//2

```

So let me just do that. I will say while the list has at least two elements, what do you mean by that. The list will have definitely all the elements that L has, but your begin and end is pointing to 0 and len L minus 1. As long as end minus begin is greater than, let say 0, which is it has at least one element or let us say it has at least two elements, that will be easy.

We will handle the case when the number of elements is less than or equal to 1. You see, we can always handle that separately. We will only look at the case where n minus begin is greater than 1, which means it has at least two elements. So, what do you do? I calculate the middle, which is, what is middle, begin plus end divided by 2. Remember, we discussed that, it is an integral part of begin plus end by 2 that is what you mean by middle

Let me patiently think. What now, I have computed my middle, right now, I will check the mid element with the given element k. If this is greater than k, if this is greater than k, then k should be on this side. As simple as that. So, let me write that down, that is interesting you see.

(Refer Slide Time: 13:56)

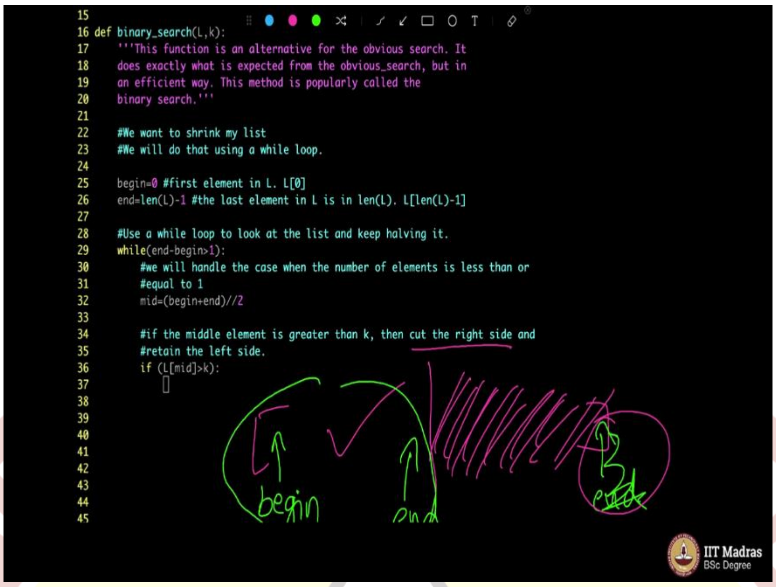


```python
def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1
        mid=(begin+end)//2

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
        if (L[mid]>k):
```

So, if it is greater than k, let us say if my L of mid, mid element is greater than k, what does that mean? If the middle element is greater than the given element k then right them cut the right side and retain the left side. What do I mean by this? I think I should annotate it once again. I mean, people who are, who know the code already are finding it boring, but then this code is not meant for you if you know it already. So, buzz off and then write a piece of code.

This is meant for people who are seeing it for the first time, I am just kidding. So, you have a list here you have a middle point here. I say L of mid this particular thing if it is greater than k. This is a very big number, assume your k is some 16 and this number happens to be some 120 then your 16 is definitely somewhere here, which means you must retain this part and get rid of this part. How do I do that? Retain left, get rid of right that is what I say here, as you can see, retain.

(Refer Slide Time: 15:06)

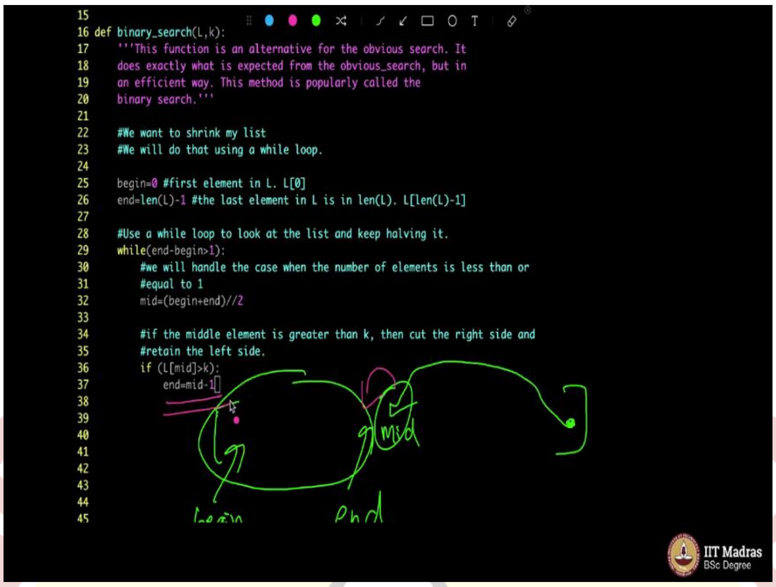


```
def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1
        mid=(begin+end)//2

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
        if (L[mid]>k):
```

Cut the right side and retain the left side. So, that is what I am going to do now. Let us see how I am going to do that. Let me keep the cursor side. So, how do you do that? I will do that by simply considering. So, what should I do? I should look at this cut the right side. Cut the right side means what? This has to be cut and this has to be retained.

This was begin and this was end, you see. One minute let me change the, this was begin, begin was a first one this was the end. So, in order for you to cut the right side and retain the left side you should simply bring this particular end here that is all, correct. This if you make it end, automatically this part is ignored. Now, we will concentrate only on this side. You see most of these things are common sensical, but when you are trying to write a code, it does not appear commonsensical.

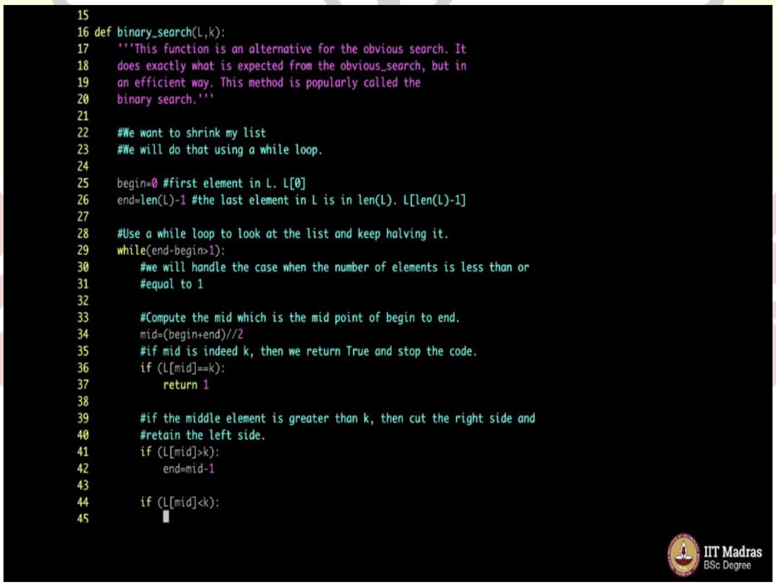
(Refer Slide Time: 16:28)



```
def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1
        mid=(begin+end)//2

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
        if (L[mid]>k):
            end=mid-1
```

So, what do I do? I will simply make my end become equal to my middle minus 1 so begin remains the same. You see, the begin part it was like this, the mid was here, the begin remains the same, but the end, which was here you bring it here, and you make end to be this, so you concentrate on this region. So, end will be if this is mid, end will be 1 before that. So, it is mid minus 1.

(Refer Slide Time: 17:00)



```
def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2
        #if mid is indeed k, then we return True and stop the code.
        if (L[mid]==k):
            return 1

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
        if (L[mid]>k):
            end=mid-1

        if (L[mid]<k):
            
```

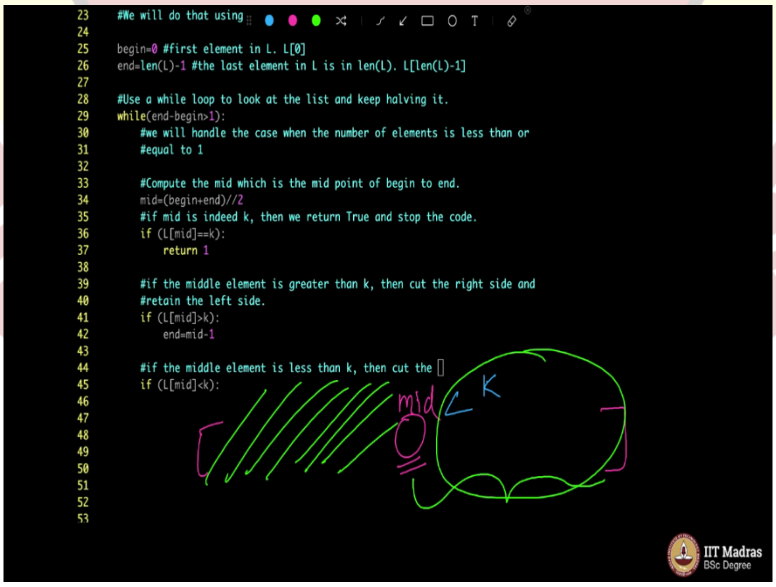
Alright. So far, so good. It is mid minus 1, but then what we should probably do is come up here and check if L of mid is indeed equal to k. As I am seeing these kind of things as you are programming you should realize, you should fix all the problems as you are programming. I just realized that you should once check if the middle is actually k. If it is k then you return 1.

Compute, wait, wait, let us stay organized. Compute, I am a little particular about grammar, so compute C capital, compute the mid, which is the midpoint of beginning to end, which is given by this formula we saw that in detail, you see. So, when there are even number of elements there can be two middle elements. So, we realize that we will call the left one as the middle. If you know what I mean. If I am, if it is confusing do not worry. I am just computing mid here that is all I want to say.

So, if L of mid is equal to k I will return 1. If this is not true, then the middle one was not k we will go ahead. We will go ahead. And then if L of mid is greater than k, then end equals mid minus 1. What if this is not true? If L of mid is less than k. I can as well put a else here, but I prefer not doing that. Sometimes it is easy on the mind to not put else. Else means, if this does not happen, whatever happens execute that somehow. As a programmer I have always realized that not using else as much as possible helps.

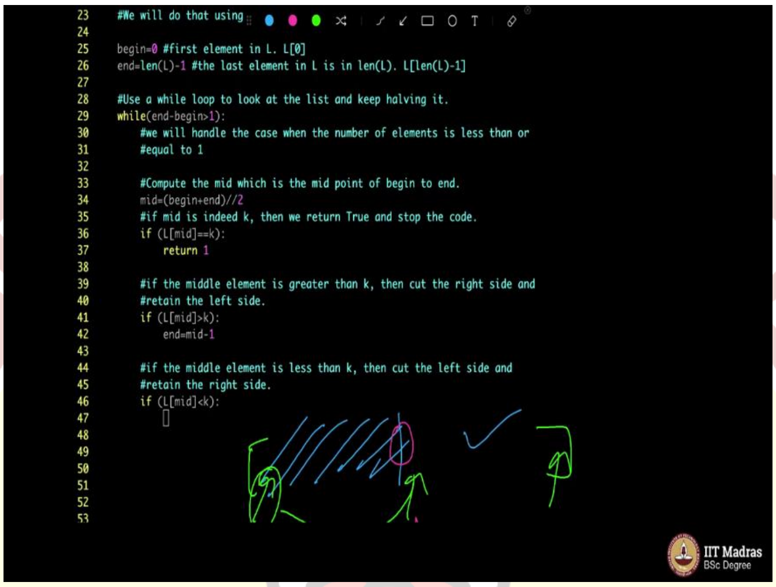
So, what have you done here? If it is equal, you have seen it that is all. If mid is indeed k rather L of mid is made is indeed k, then we return true, which is 1 and stop the code. If not, then we go ahead. If L of mid is greater than k, end equals mid minus 1. I am going really slow here, this, these two lines are very important. Just concentrate on these two lines they are very important. Once you are through you go ahead, you pause and then think.

(Refer Slide Time: 19:30)


```
#We will do that using

begin=0 #first element in L. L[0]
end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

#Use a while loop to look at the list and keep halving it.
while(end-begin>1):
    #we will handle the case when the number of elements is less than or
    #equal to 1

    #Compute the mid which is the mid point of begin to end.
    mid=(begin+end)//2
    #if mid is indeed k, then we return True and stop the code.
    if (L[mid]==k):
        return 1

    #if the middle element is greater than k, then cut the right side and
    #retain the left side.
    if (L[mid]>k):
        end=mid-1

    #if the middle element is less than k, then cut the 
    if (L[mid]<k):
```

Similarly, if L of mid is less than k, some of you are not following what I am doing. So, let me write the comment once again here. If the middle element is less than k then cut the which side, because it is right here you just cannot say it is left here. It should be should be there in your mind, so let us work it out. Again, the same old thing. I mean, I can do repeated chanting of this very concept. It is all worth it, because it should be easier in your mind. So, this mid here this mid, L of mid this very element if it is less than k, if it is less than k then your k, if it is there, it will be on the right side of this, then what you do you discard the left side and you retain the right side. So, discard left retain right, exactly the opposite of before.

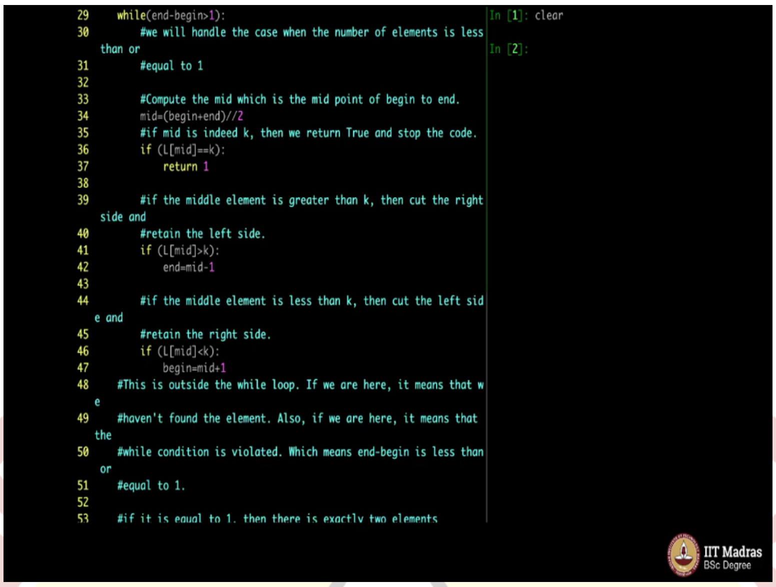
(Refer Slide Time: 20:41)



```
#We will do that using

begin=0 #first element in L. L[0]
end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

#Use a while loop to look at the list and keep halving it.
while(end-begin>1):
    #we will handle the case when the number of elements is less than or
    #equal to 1

    #Compute the mid which is the mid point of begin to end.
    mid=(begin+end)//2
    #if mid is indeed k, then we return True and stop the code.
    if (L[mid]==k):
        return 1

    #if the middle element is greater than k, then cut the right side and
    #retain the left side.
    if (L[mid]>k):
        end=mid-1

    #if the middle element is less than k, then cut the left side and
    #retain the right side.
    if (L[mid]<k):
        
```

If the middle element is less than k, then cut the left side and retain the right side. Comment this. Of course, you can put a triple quote too, but I do not want to do it, I like hash. So, what do you do? You just in case L of mid is less than k, you retain the right side and cut the left side. What do you mean by that? What do you mean by retaining the, cut the left and retaining the right? It simply means that this was begin this was end, bring this begin here. Make begin immediately after your mid, this was your mid. You see, remember, this was the mid. Mid plus 1 will be your begin, that is all I need to type here.

(Refer Slide Time: 21:41)

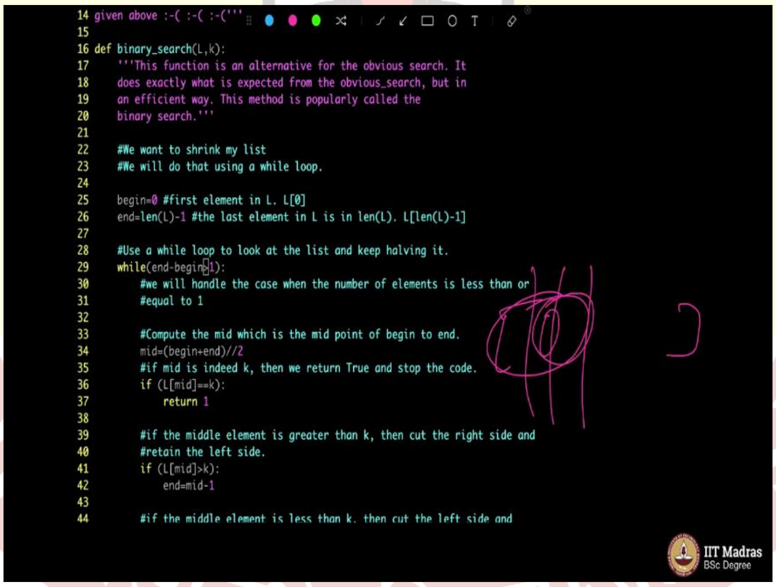


```
    while(end-begin>1):
        #we will handle the case when the number of elements is less
 than or
        #equal to 1

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2
        #if mid is indeed k, then we return True and stop the code.
        if (L[mid]==k):
            return 1

        #if the middle element is greater than k, then cut the right
 side and
        #retain the left side.
        if (L[mid]>k):
            end=mid-1

        #if the middle element is less than k, then cut the left sid
e and
        #retain the right side.
        if (L[mid]<k):
            begin=mid+1
    #This is outside the while loop. If we are here, it means that w
e
    #haven't found the element. Also, if we are here, it means that
the
    #while condition is violated. Which means end-begin is less than
 or
    #equal to 1.

    #if it is equal to 1, then there is exactly two elements
```

```
In [1]: clear

In [2]:
```

Let me type that here now. So, your begin simply will be mid plus 1, that is it. The code is almost over. So, let me just see what has happened here.

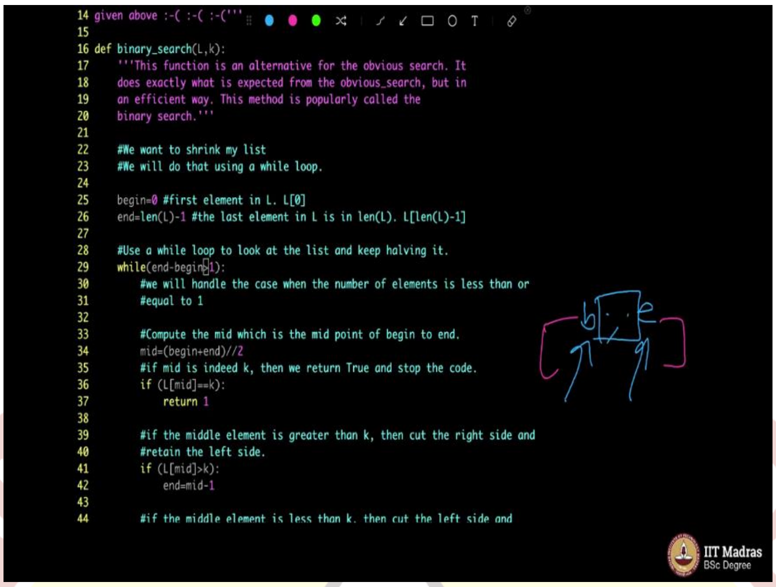
(Refer Slide Time: 21:58)



```
given above :-( :-( :-('''

def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2
        #if mid is indeed k, then we return True and stop the code.
        if (L[mid]==k):
            return 1

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
        if (L[mid]>k):
            end=mid-1

        #if the middle element is less than k, then cut the left side and
```

Let me think aloud binary search, you take a list L and search element k. What do I do? The beginning element is 0, the ending element is Len of L minus 1 the last index basically have a list L. And then while, what does it mean? Use a while loop to look at the list and keep halving it. I am looking at the list. End minus begin is greater than 1, the list keeps shrinking, you see, it keeps shrinking, it keeps shrinking, it keeps shrinking.

(Refer Slide Time: 22:29)

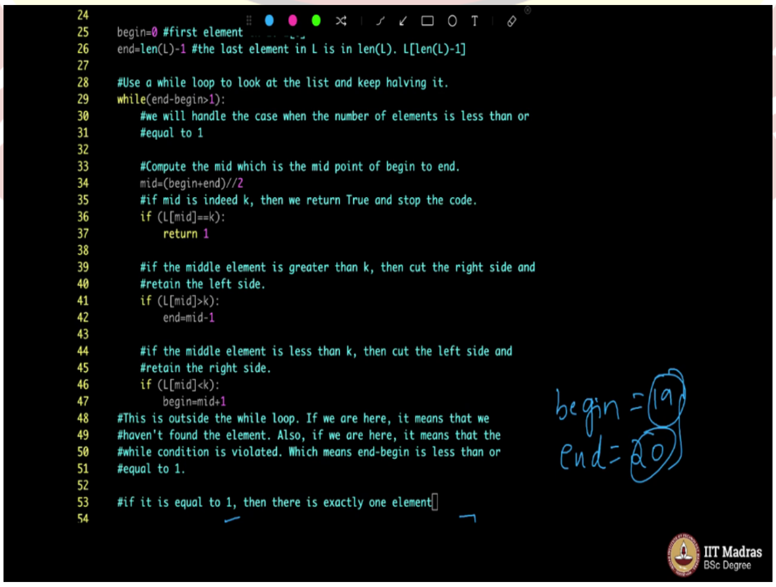


```
given above :-( :-( :-('''

def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2
        #if mid is indeed k, then we return True and stop the code.
        if (L[mid]==k):
            return 1

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
        if (L[mid]>k):
            end=mid-1

        #if the middle element is less than k, then cut the left side and
```

Whatever it is, whatever is the list. Whatever is the list. What I want to do is, I want to see if begin and end wherever that is being pointed. Begin and end, this region is my region of interest always. That should have at least two elements, at least two elements. If not, it is not interesting, is all I am saying here. End minus begin is greater than 1, good.

So, and then I compute mid, mid is simply the midpoint of that region of interest. Region of interest basically is from beginning to end. And just in case, the entry in the middle is k, then your job is done, you leave home. You are done, you have found the element and you return 1. If not, then the L of mid middle element is greater than k or is less than k, correct?

(Refer Slide Time: 23:26)



```
    begin=0 #first element
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2
        #if mid is indeed k, then we return True and stop the code.
        if (L[mid]==k):
            return 1

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
        if (L[mid]>k):
            end=mid-1

        #if the middle element is less than k, then cut the left side and
        #retain the right side.
        if (L[mid]<k):
            begin=mid+1
    #This is outside the while loop. If we are here, it means that we
    #haven't found the element. Also, if we are here, it means that the
    #while condition is violated. Which means end-begin is less than or
    #equal to 1.

    #if it is equal to 1, then there is exactly one element
```

Let me scroll down, it is greater than k or is less than k? If it is greater than k what do you do? If it is greater than k, all you do is, you pull the end make end come to the middle. That is because your region of interest then will be the first half not the second half. So, then you cut the right side and retain the left side. If the middle element is less than k, which means, as I told you, I am not going to write it once again, I wrote it some dozen number of times I cut the left hand retain the right think about it, and that is done.

How do you cut the left side, you cut the left side by making begin, bringing begin from this end to the middle. So, middle one definitely it is not there we have checked it. If we are outside this it means that L of mid was not k, so I start with mid plus 1, begin is mid plus 1. Rehearse this properly, think about it, and only after understanding this will go ahead.

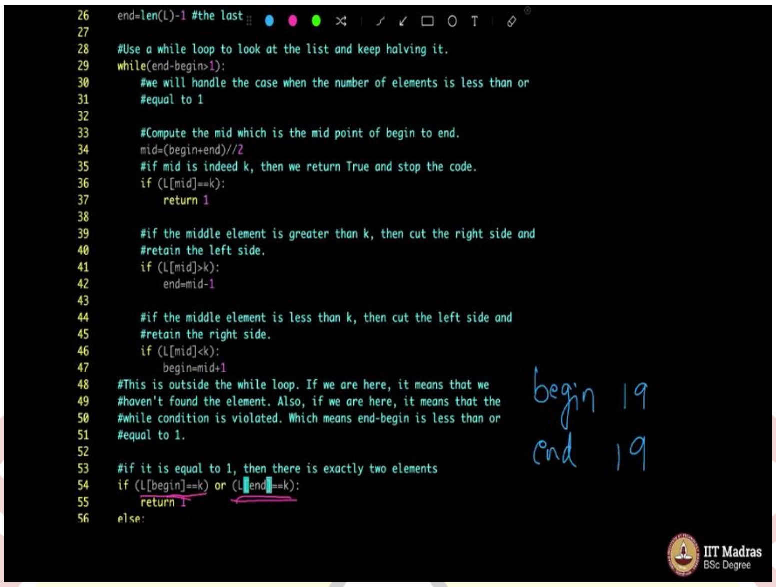
So, what next? I come out of the while loop. This is outside, this is outside the while loop. If we are here, it means that we have not found the element. Also, if we are here, it means that the while condition is violated, which means, what does it mean, end minus begin, it means end minus begin is not greater than 1, it is equal to 1 and that is when you come out or it can be less than 1. Which means, end minus begin is either 1 or 0 or minus 1 or minus 2 or whatever. It is less than or equal to 1.

What if it is equal to 1? Let us just check. Now, if it is equal to 1. What is equal to 1? Let me just think. What am I doing here? I am saying, if you are outside the while loop, it means that end minus begin is less than or equal to 1. Let us say it is equal to 1, what does it mean if it is equal to 1. It means that begin equals end.

It means begin equals end. Only then end minus begin, which it means if it is equal to 1, what am I saying? If it is equal to 1, then there is exactly one element in the region of interest. Let me illustrate that. See, we all get confused, so it is okay to be confused, but then the point is we have to clarify our confusion and move towards clarification.

So, if begin end minus begin is equal to 1, then let us say end is some 20 and begin is some 19, only then end minus begin will be 1, which means the region of interest starts here and ends here. So, simply there are two elements 19th element 20th element, only two elements are there.

(Refer Slide Time: 27:20)



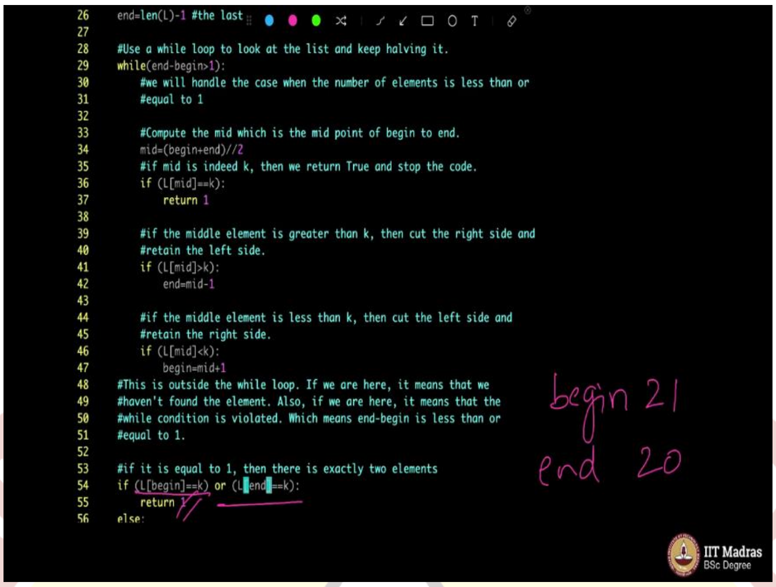
```
end=len(L)-1 #the last

#Use a while loop to look at the list and keep halving it.
while(end-begin>1):
    #we will handle the case when the number of elements is less than or
    #equal to 1

    #Compute the mid which is the mid point of begin to end.
    mid=(begin+end)//2
    #if mid is indeed k, then we return True and stop the code.
    if (L[mid]==k):
        return 1

    #if the middle element is greater than k, then cut the right side and
    #retain the left side.
    if (L[mid]>k):
        end=mid-1

    #if the middle element is less than k, then cut the left side and
    #retain the right side.
    if (L[mid]<k):
        begin=mid+1
#This is outside the while loop. If we are here, it means that we
#haven't found the element. Also, if we are here, it means that the
#while condition is violated. Which means end-begin is less than or
#equal to 1.

#if it is equal to 1, then there is exactly two elements
if (L[begin]==k) or (L[end]==k):
    return 1
else:
```

So, in that case, what should I do? If it is equal to 1 when then there is exactly two elements, I am sorry. There are exactly two elements. Then I check if L of begin is equal to k or L of end is equal to k, return 1, else return 0. So, if we are here, we have the region of interest with only two elements, and we check those two elements. Either the beginning part should be, because only there is a begin and end and the end is actually one more than begin.

I will check if L of begin is k. If it is true, then I return 1 or if L of end is k return 1. If it is equal to one, then this happens. What if it is not equal to 1? What if it is equal to 0? Still this condition will check for L of begin if end minus begin is equal to 0, then begin will be equal to end do you see, think about it for some time. All the cases that this covers.

So, what if begin and end is such that, one moment. Begin is 19, I said end is 20 then the difference is 1, it comes outside the loop then I check L of 19 and L of 20. Fine. But what if begin is 19, and end is also 19. In that case I will check L of begin once and L of end once. L of 19, L of 19 I will check it is okay, it is written then but still the code works. In either case, it will be one if it is equal to k, and I will return 1. And what if there could be a situation where begin can become more than end, that can happen actually. Anyways, if it happens or if it does not happen, I do not care.

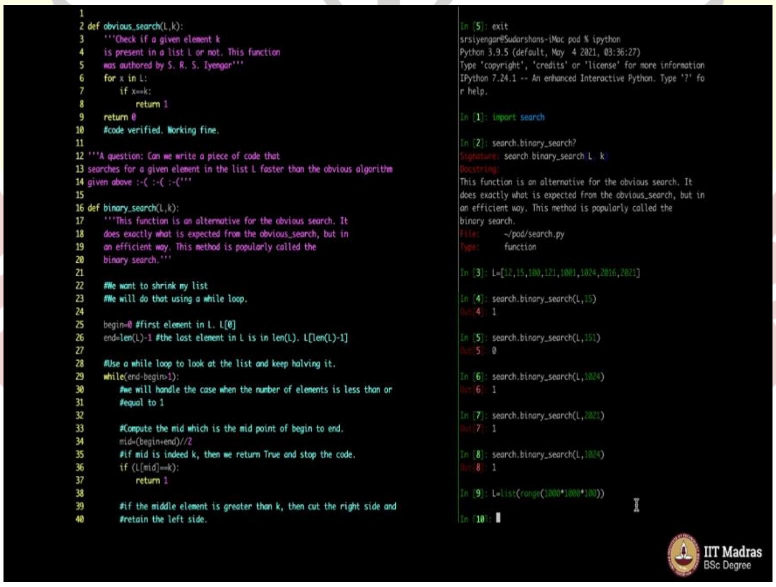
(Refer Slide Time: 29:18)



```python
    end=len(L)-1 #the last

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2
        #if mid is indeed k, then we return True and stop the code.
        if (L[mid]==k):
            return 1

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
        if (L[mid]>k):
            end=mid-1

        #if the middle element is less than k, then cut the left side and
        #retain the right side.
        if (L[mid]<k):
            begin=mid+1
    #This is outside the while loop. If we are here, it means that we
    #haven't found the element. Also, if we are here, it means that the
    #while condition is violated. Which means end-begin is less than or
    #equal to 1.

    #if it is equal to 1, then there is exactly two elements
    if (L[begin]==k) or (L[end]==k):
        return 1
    else:
```

Just in case that happens assuming that my begin is greater and end is smaller, even in that case, I will check L of begin and L of end and then return 1, just if it is present, otherwise, I will return 0. Perfect. So, let me now go ahead and try to execute this. What I will now do is try zooming this out, just so that it is all visible in one screen. The font may appear a little smaller, please do not mind.

(Refer Slide Time: 30:01)



```python

def obvious_search(L,k):
    '''Check if a given element k
    is present in a list L or not. This function
    was authored by S. R. S. Iyengar'''
    for x in L:
        if x==k:
            return 1
    return 0
    #code verified. Working fine.

'''A question: Can we write a piece of code that
searches for a given element in the list L faster than the obvious algorithm
given above :-( :-( :-('''

def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2
        #if mid is indeed k, then we return True and stop the code.
        if (L[mid]==k):
            return 1

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
```

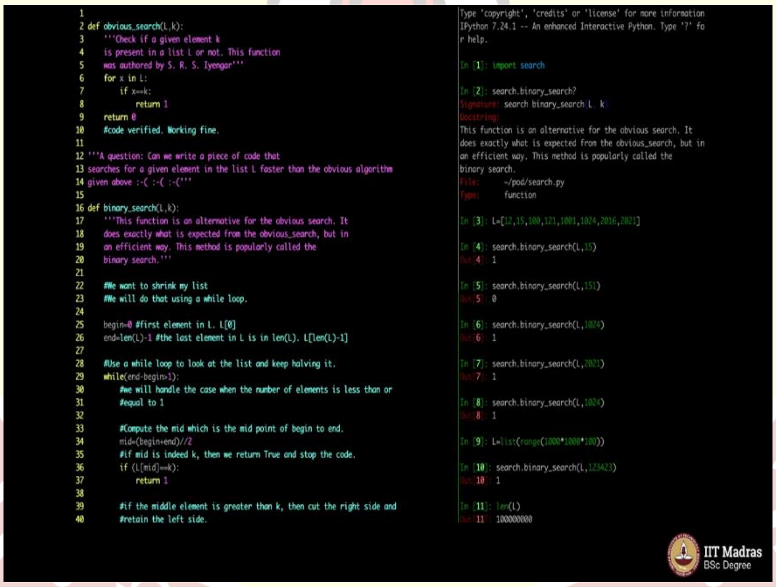
```
In [5]: exit
srsiyengar@Sudarshans-iMac pod % ipython
Python 3.9.5 (default, May  4 2021, 03:36:27)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.24.1 -- An enhanced Interactive Python. Type '?' fo
r help.

In [1]: import search

In [2]: search.binary_search?
Signature: search binary_search(L, k)
Docstring:
This function is an alternative for the obvious search. It
does exactly what is expected from the obvious_search, but in
an efficient way. This method is popularly called the
binary search.
File:      ~/pod/search.py
Type:      function

In [3]: L=[12,15,100,121,1001,1024,2016,2021]

In [4]: search.binary_search(L,15)
Out[4]: 1

In [5]: search.binary_search(L,151)
Out[5]: 0

In [6]: search.binary_search(L,1024)
Out[6]: 1

In [7]: search.binary_search(L,2021)
Out[7]: 1

In [8]: search.binary_search(L,1024)
Out[8]: 1

In [9]: L=list(range(1000*1000*100))

In [10]: 
```

Say I am going to restart this then say import search, and then search. You see, one function is obvious search, one function is binary search. Question mark it will give you the details of this function, the function is an alternative for etc, etc. Good. So, I can search for binary search in a list L, so let me just first create the list L. What was my list L? L is, L can be anything, let us say 12, 15, 100, 121, 1001, 1024, 2016, 2021 which is this year. So, now I can try to search an element here.

Very good. It did not throw any error. I mean, there was no problem. Absolutely, in fact, it got executed. I thought there would at least be some logical error. It is working perfectly fine. 151 is not there, so it says no 1024 is there. Let me enable cursor, 1024 is there and so it says 1024 here. And let me choose 2021, 2021 and it shows 1. Perfect, but the list is very small. I want the list to be very big. So, what I will do is list of range of this is 1 million, thousand times thousands is 1 million times 100 is 100 million.

This is easy. You see, I meant to create a big list that is sorted. In fact, you can create a big list that is not 0,1, 2, 3, 4, 5 up to 100 million minus 1 of course, which is not, which is sorted, you can create some random numbers there, but that is a lot of headaches I will not do it. So, a program that works for this should work for anything else.

(Refer Slide Time: 32:14)

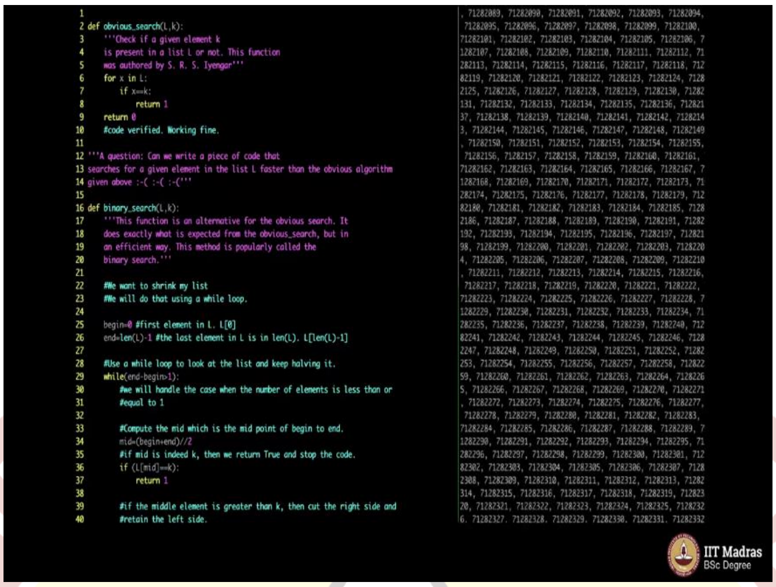


```

def obvious_search(L,k):
    '''Check if a given element k
    is present in a list L or not. This function
    was authored by S. R. S. Iyengar'''
    for x in L:
        if x==k:
            return 1
    return 0
    #code verified. Working fine.

'''A question: Can we write a piece of code that
searches for a given element in the list L faster than the obvious algorithm
given above :-( :-( :-('''

def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2
        #if mid is indeed k, then we return True and stop the code.
        if (L[mid]==k):
            return 1

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
```

```
Type 'copyright', 'credits' or 'license' for more information
IPython 7.24.1 -- An enhanced Interactive Python. Type '?' fo
r help.

In [1]: import search

In [2]: search.binary_search?
Signature: search binary_search(L, k)
Docstring:
This function is an alternative for the obvious search. It
does exactly what is expected from the obvious_search, but in
an efficient way. This method is popularly called the
binary search.
File:      ~/pod/search.py
Type:      function

In [3]: L=[12,15,100,121,1001,1024,2016,2021]

In [4]: search.binary_search(L,15)
Out[4]: 1

In [5]: search.binary_search(L,151)
Out[5]: 0

In [6]: search.binary_search(L,1024)
Out[6]: 1

In [7]: search.binary_search(L,2021)
Out[7]: 1

In [8]: search.binary_search(L,1024)
Out[8]: 1

In [9]: L=list(range(1000*1000*100))

In [10]: search.binary_search(L,123423)
Out[10]: 1

In [11]: len(L)
Out[11]: 100000000
```

So let me now say search binary. On L, let me try finding a random number like this. It should be there because why what is L? You will see L is all numbers from 0 to 100 million. Let me print L and show it you. It will go on and on for the entire day. It will even take some time to start. Oh god, why did I even execute this code?

(Refer Slide Time: 32:38)

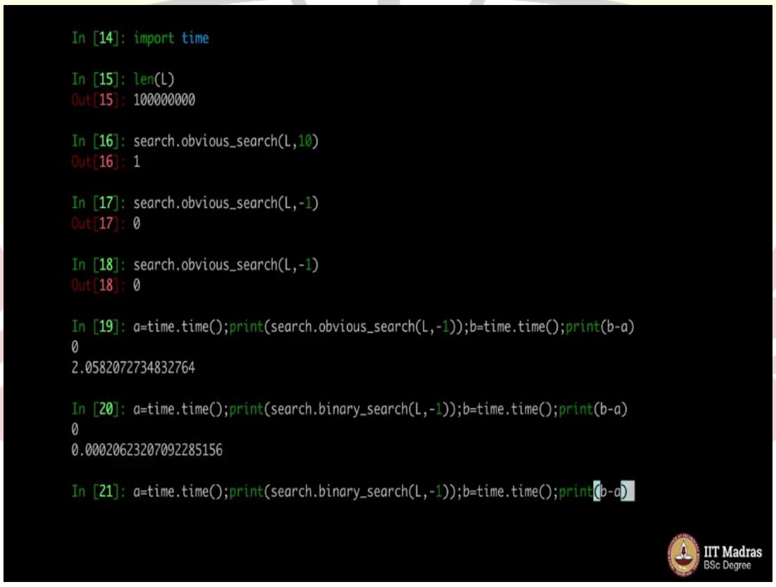


```python

def obvious_search(L,k):
    '''Check if a given element k
    is present in a list L or not. This function
    was authored by S. R. S. Iyengar'''
    for x in L:
        if x==k:
            return 1
    return 0
    #code verified. Working fine.

'''A question: Can we write a piece of code that
searches for a given element in the list L faster than the obvious algorithm
given above :-( :-( :-('''

def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
    end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

    #Use a while loop to look at the list and keep halving it.
    while(end-begin>1):
        #we will handle the case when the number of elements is less than or
        #equal to 1

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2
        #if mid is indeed k, then we return True and stop the code.
        if (L[mid]==k):
            return 1

        #if the middle element is greater than k, then cut the right side and
        #retain the left side.
```

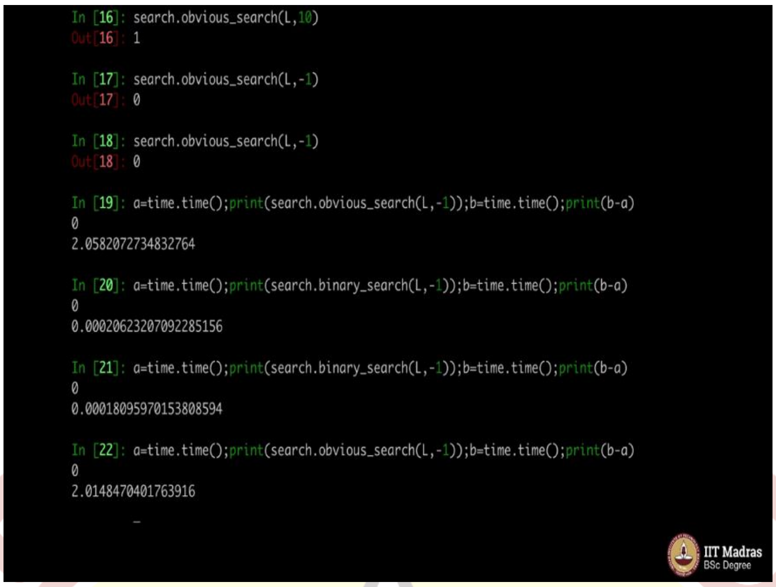
```
, 71282089, 71282090, 71282091, 71282092, 71282093, 71282094,
 71282095, 71282096, 71282097, 71282098, 71282099, 71282100,
71282101, 71282102, 71282103, 71282104, 71282105, 71282106, 7
1282107, 71282108, 71282109, 71282110, 71282111, 71282112, 71
282113, 71282114, 71282115, 71282116, 71282117, 71282118, 712
82119, 71282120, 71282121, 71282122, 71282123, 71282124, 7128
2125, 71282126, 71282127, 71282128, 71282129, 71282130, 71282
131, 71282132, 71282133, 71282134, 71282135, 71282136, 712821
37, 71282138, 71282139, 71282140, 71282141, 71282142, 7128214
3, 71282144, 71282145, 71282146, 71282147, 71282148, 71282149
, 71282150, 71282151, 71282152, 71282153, 71282154, 71282155,
 71282156, 71282157, 71282158, 71282159, 71282160, 71282161,
71282162, 71282163, 71282164, 71282165, 71282166, 71282167, 7
1282168, 71282169, 71282170, 71282171, 71282172, 71282173, 71
282174, 71282175, 71282176, 71282177, 71282178, 71282179, 712
82180, 71282181, 71282182, 71282183, 71282184, 71282185, 7128
2186, 71282187, 71282188, 71282189, 71282190, 71282191, 71282
192, 71282193, 71282194, 71282195, 71282196, 71282197, 712821
98, 71282199, 71282200, 71282201, 71282202, 71282203, 7128220
4, 71282205, 71282206, 71282207, 71282208, 71282209, 71282210
, 71282211, 71282212, 71282213, 71282214, 71282215, 71282216,
 71282217, 71282218, 71282219, 71282220, 71282221, 71282222,
71282223, 71282224, 71282225, 71282226, 71282227, 71282228, 7
1282229, 71282230, 71282231, 71282232, 71282233, 71282234, 71
282235, 71282236, 71282237, 71282238, 71282239, 71282240, 712
82241, 71282242, 71282243, 71282244, 71282245, 71282246, 7128
2247, 71282248, 71282249, 71282250, 71282251, 71282252, 71282
253, 71282254, 71282255, 71282256, 71282257, 71282258, 712822
59, 71282260, 71282261, 71282262, 71282263, 71282264, 7128226
5, 71282266, 71282267, 71282268, 71282269, 71282270, 71282271
, 71282272, 71282273, 71282274, 71282275, 71282276, 71282277,
 71282278, 71282279, 71282280, 71282281, 71282282, 71282283,
71282284, 71282285, 71282286, 71282287, 71282288, 71282289, 7
1282290, 71282291, 71282292, 71282293, 71282294, 71282295, 71
282296, 71282297, 71282298, 71282299, 71282300, 71282301, 712
82302, 71282303, 71282304, 71282305, 71282306, 71282307, 7128
2308, 71282309, 71282310, 71282311, 71282312, 71282313, 71282
314, 71282315, 71282316, 71282317, 71282318, 71282319, 712823
20, 71282321, 71282322, 71282323, 71282324, 71282325, 7128232
6, 71282327, 71282328, 71282329, 71282330, 71282331, 71282332
```

Alright. It will go on and on and on. Anyways, meanwhile, let us come this side and try to see that we have obvious search here, and we have binary search here. Correct? obvious search took some time. We use that import time function and try doing something. Let us try doing the same thing for binary searched and see if it.

(Refer Slide Time: 33:14)



```
In [14]: import time

In [15]: len(L)
Out[15]: 100000000

In [16]: search.obvious_search(L,10)
Out[16]: 1

In [17]: search.obvious_search(L,-1)
Out[17]: 0

In [18]: search.obvious_search(L,-1)
Out[18]: 0

In [19]: a=time.time();print(search.obvious_search(L,-1));b=time.time();print(b-a)
0
2.0582072734832764

In [20]: a=time.time();print(search.binary_search(L,-1));b=time.time();print(b-a)
0
0.00020623207092285156

In [21]: a=time.time();print(search.binary_search(L,-1));b=time.time();print(b-a)
```



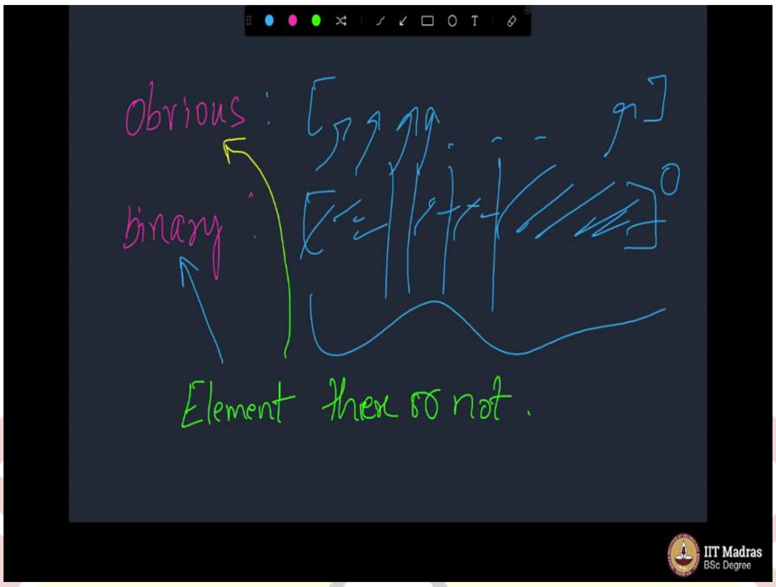
```
In [16]: search.obvious_search(L,10)
Out[16]: 1

In [17]: search.obvious_search(L,-1)
Out[17]: 0

In [18]: search.obvious_search(L,-1)
Out[18]: 0

In [19]: a=time.time();print(search.obvious_search(L,-1));b=time.time();print(b-a)
0
2.0582072734832764

In [20]: a=time.time();print(search.binary_search(L,-1));b=time.time();print(b-a)
0
0.00020623207092285156

In [21]: a=time.time();print(search.binary_search(L,-1));b=time.time();print(b-a)
0
0.00018095970153808594

In [22]: a=time.time();print(search.obvious_search(L,-1));b=time.time();print(b-a)
0
2.0148470401763916

         _
```

I think our friend will not stop. So, I will cancel it. There is a keyboard interrupt that you can do by pressing Ctrl C on your system that is what I did, and let me clear the screen, and then say import time. What I can do is maybe, I will try to zoom this increase the font and my L is already there. In search, if I say obvious search in the list L, I want to find the number 10, it does very quickly, but what if it is a very big number, you see.

Maybe something that is not even here in the list, then it takes a long time. We saw that, it was quite some time here. How did we do that? We did the following. We said input time. What was that code, a equals time, time correct, and then we did print search, obvious search L minus 1, something that is not there in the list and then b equals time, time, I hope I am writing the same thing that we wrote long back then I will say print b minus a. The time after minus time before this should give me time that it takes.

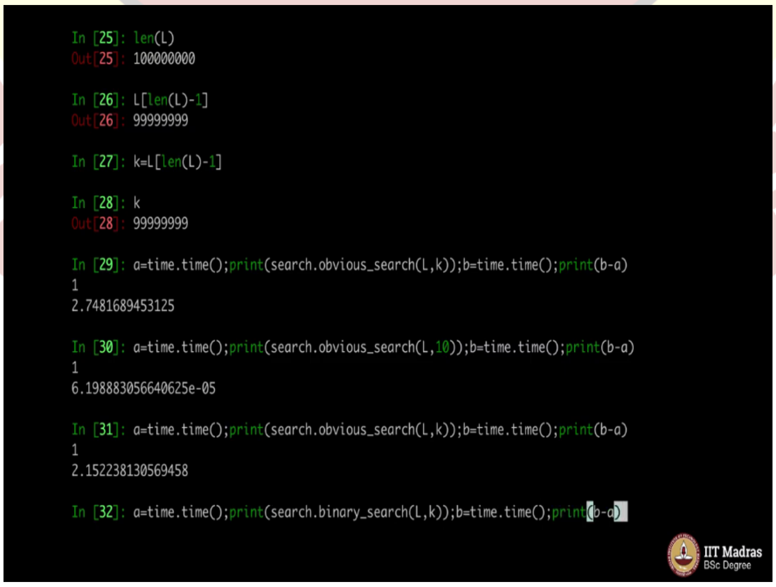
It was how many some? So, it takes two seconds for this. So, instead let me do this for the binary search. Let us see how much time it takes. Did you see the magic? It did not take any time. As I executed it, boom, it comes. Whereas, for obvious you see, this has been easy, obvious search if I am executing, it is taking time. That is because why is this happening?

(Refer Slide Time: 35:13)



That is because, one second, that is because in the obvious search, what is happening is the following it goes through the entire list, is it here? Is it here, is it here, is it here, goes until the end, it is not here and then declared 0, but then in binary search, the halving principle, you see, dictionary searching, give up on this, give up on this, give up on this, give up on this quickly the list becomes small and very quickly we decide if the element is there or not, is quickly decided in case of what in case of binary search, but in case of the obvious search, it takes a long time, we just saw that.

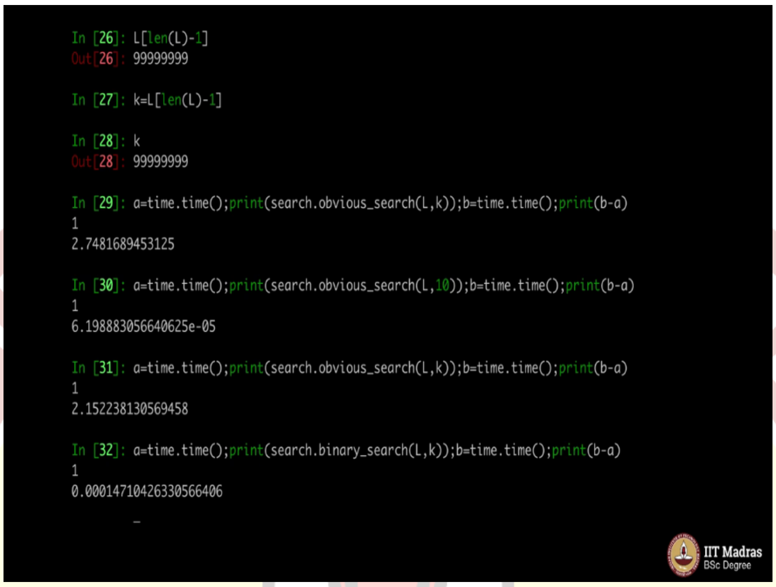
(Refer Slide Time: 36:09)



```
In [25]: len(L)
Out[25]: 100000000

In [26]: L[len(L)-1]
Out[26]: 99999999

In [27]: k=L[len(L)-1]

In [28]: k
Out[28]: 99999999

In [29]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
1
2.7481689453125

In [30]: a=time.time();print(search.obvious_search(L,10));b=time.time();print(b-a)
1
6.198883056640625e-05

In [31]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
1
2.152238130569458

In [32]: a=time.time();print(search.binary_search(L,k));b=time.time();print(b-a)
```

Let us play around just a little more. Let me just remove this. So, what if I want to do the obvious search of some element, which is there in the list? What is that, which is I think, wait a minute. What was the, let us fix a k so that we put the same k and then check it. Let me. So, what I will do is k equals, what was the number, L is how much length of L is so much, let me clear the screen for your people.

(Refer Slide Time: 36:55)



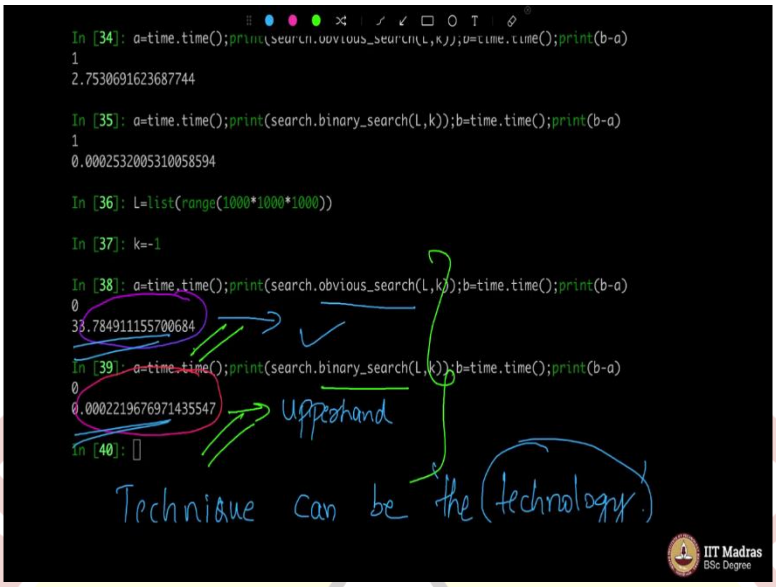
```
In [26]: L[len(L)-1]
Out[26]: 99999999

In [27]: k=L[len(L)-1]

In [28]: k
Out[28]: 99999999

In [29]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
1
2.7481689453125

In [30]: a=time.time();print(search.obvious_search(L,10));b=time.time();print(b-a)
1
6.198883056640625e-05

In [31]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
1
2.152238130569458

In [32]: a=time.time();print(search.binary_search(L,k));b=time.time();print(b-a)
1
0.00014710426330566406

        _
```

So, length of L is so much, which means the last element of L will be so much. I will call this as k, len of L minus 1, the last element, which means this element, whatever is in k, whatever k is denoting that number is actually there in the list. Now, let us do the obvious search, and try to find not minus 1, but k here. I hope you all know what I am doing.

Let us find k here and it takes time because it should go to the end of the list. But if you were to find instead of k, which is 999, 999, 99 if it has some 10, it will execute in no time. It is what 0.0006 seconds because it is in the first part of L, even obvious such works there. Because is L of 0, 10 is L of 1, 10 is L of 2, 10 so on up to L of 10, 10 very quickly executed.

Even binary search will be fast here, obvious search will be fast here. But then if you put a k here, we saw k or minus 1 here are a number which will make your obvious search go through the entire list, it takes some time it takes some non-trivial time you see. But then let us try I am just going by the history asparagus of the history a binary search for k only L comma k, you see that? How it is really, really fast. Let us see more magic now.

(Refer Slide Time: 38:29)



```
In [34]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
1
2.7530691623687744

In [35]: a=time.time();print(search.binary_search(L,k));b=time.time();print(b-a)
1
0.0002532005310058594

In [36]: L=list(range(1000*1000*1000))

In [37]: k=-1

In [38]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
0
33.784911155700684

In [39]: a=time.time();print(search.binary_search(L,k));b=time.time();print(b-a)
0
0.00022196769714355547

In [40]: 
```

What did we do? Let me repeat it. Obvious search took 2.7 seconds and binary search took no time 0.0002 seconds. Now, what I will do, I will write L as list in the range, make it 1 billion instead of 100 million. I must warn you this is going to take some time. Anyways, I believe, I do not know whether the editing team has edited the time that it takes in between.

If they have not edited it is good because you people will get to know how much time it actually took to create the list also is not so easy. 1 billion entries it is populating with 0, 1, 2, 3 up to 1 billion. It will take few seconds, a few seconds of the order of I think, it took some 20 seconds here. Now, let me say k is equal to minus 1, which is not here. Of course, it is not being displayed because my keyboard is not working because the system is hung.

The monitor is not displaying anything, it will take some time that is because what you are doing here is very expensive. You are using the whole of your memory for this list. If you open your activity, I mean whatever you if you are on Linux, you can try to see your memory resources. If you are sort of a geek, you know how to do it. It will show that you are using all your it is called RAM memory for this. I hope it comes out.

Let us wait, let us wait, let us wait. Because this is really elegant, very beautiful, it is worth waiting, k equals minus 1. Good, it took so much time for this one line to come and that is because it was stuck basically. Now, let me go ahead and execute this binary search I do not know obvious search on this, and see the time that it takes. We got to be patient, it may take around a minute's time, if I am not wrong. Let us see. Waiting, waiting, waiting, waiting, waiting.

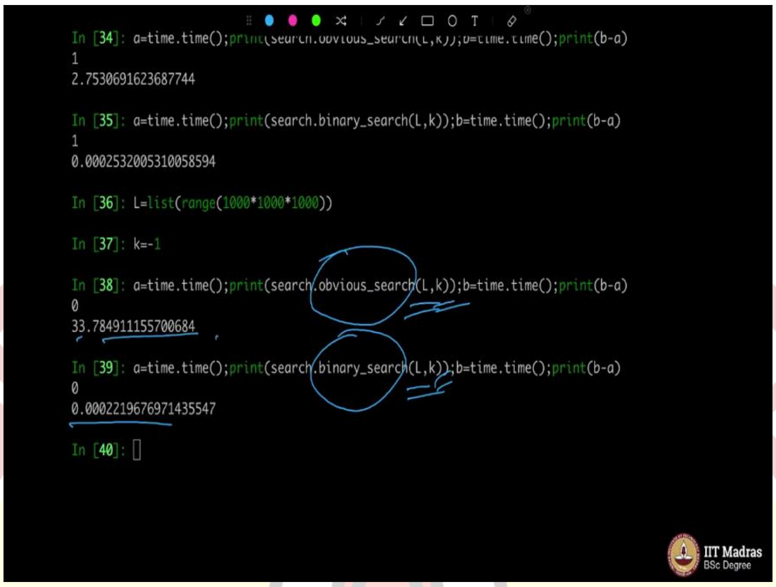
There is some way in which we can see a time in this shell because this is the time right now. Of course, this is of no use because I am computing. It take to 22:45, 10:45 PM it is. It is taking its own time. Let us wait, 33 seconds. Please observe we are in the climax point of today's discussion. It took 33 seconds for us to run the obvious search on a list with how many elements 1 billion elements where k was minus 1 which is it did not belong to the list, and it should obviously say 0 because it did not belong.

Now, what next, we will try our binary search chance. What was that? Do you see, where is 33 seconds and where is 0.0002 seconds. I mean, Mollen mountain difference. Do you observe that? Mollen mountain difference between this obvious search and the binary search. Super duper difference you cannot even believe, 33 seconds is close to half a minute. But note, let me just write this down, this is really nice to think for a minute, sort of let us mediate on this, this was your obvious search and this is was your binary search.

Point to note, both of them used your computer, this very same computer it used, my very computer. But this one used a technique, this one used some other technique. And this technique had an upper hand over this technique. This had a upper hand over this technique. What does it mean? It simply means that the technique you use to search can it itself be, can be the technology.

An English expert who sees this sentence will say what is he talking. Technique is a technology it seems. Of course, technology is derived from technique. The study of technique is called technology. By technology I mean, the hardware technology. I am using a super-duper computer, but it was a very same super duper the technique that I used matters. 33 seconds and then 0.0002 seconds. Same list, you see.

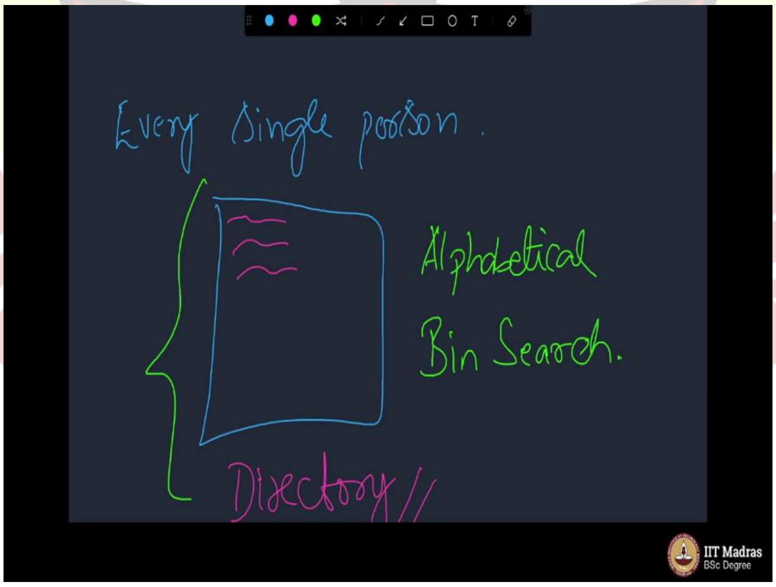
(Refer Slide Time: 43:41)



```
In [34]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
1
2.7530691623687744

In [35]: a=time.time();print(search.binary_search(L,k));b=time.time();print(b-a)
1
0.0002532005310058594

In [36]: L=list(range(1000*1000*1000))

In [37]: k=-1

In [38]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
0
33.784911155700684

In [39]: a=time.time();print(search.binary_search(L,k));b=time.time();print(b-a)
0
0.00022196769714355547

In [40]: 
```

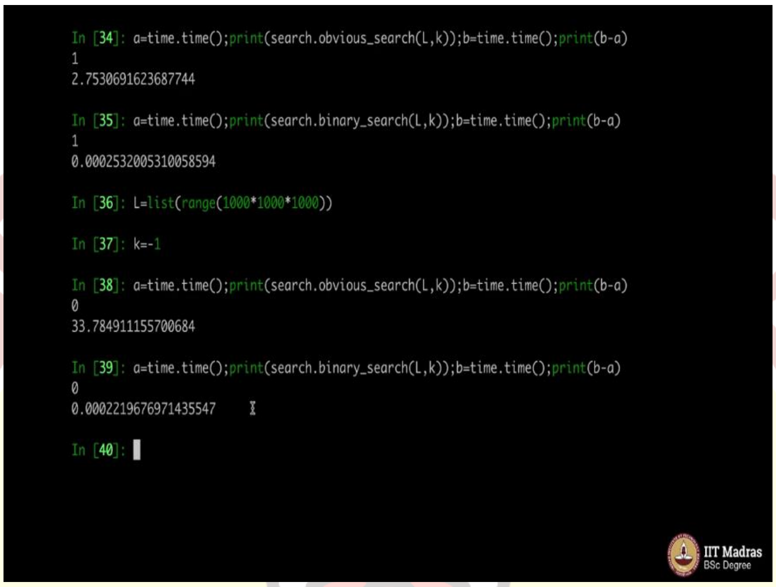
Same list, same key element, but the method that I am using is different. And that is what is giving this significant difference. In fact, very beautiful thing to observe here is no matter how big the list is binary search is super duper fast.

(Refer Slide Time: 44:15)



In fact, let me just make a note of this, it is a wonderful fact that if you were to list everyone in the world, every single person ever lived in the world. Create a telephone directory for that person with all the entries. And if you keep it in alphabetical order if this is in alphabetical order. What am I saying? I am saying, that you take every single person ever lived in the world and then write create telephone directory of this of all these people searching that will not be very difficult. Searching is very easy, if you use the binary search technique, whatever we illustrated right now. No matter how big the list is, how big the repository is binary search really rocks.

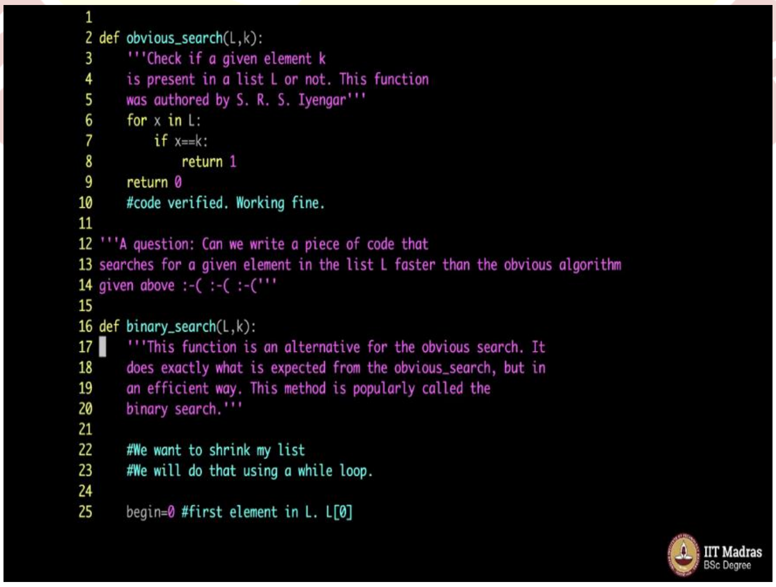
(Refer Slide Time: 45:09)



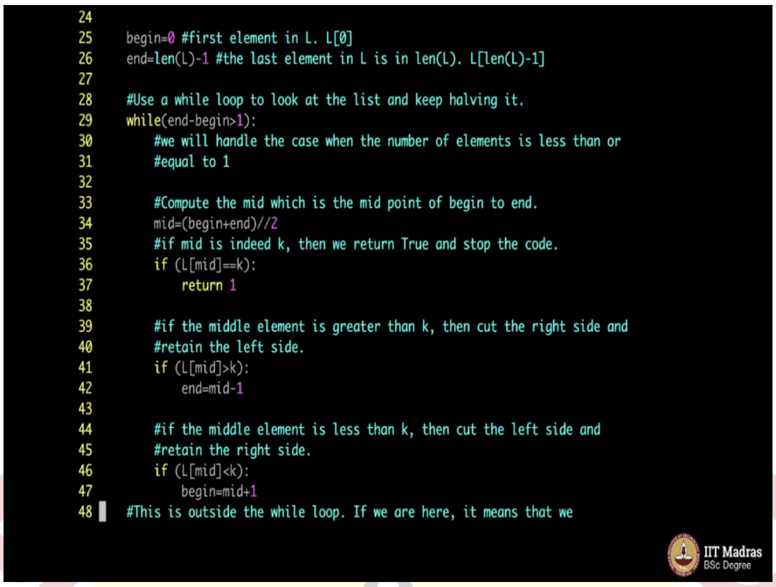
```
In [34]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
1
2.7530691623687744

In [35]: a=time.time();print(search.binary_search(L,k));b=time.time();print(b-a)
1
0.0002532005310058594

In [36]: L=list(range(1000*1000*1000))

In [37]: k=-1

In [38]: a=time.time();print(search.obvious_search(L,k));b=time.time();print(b-a)
0
33.784911155700684

In [39]: a=time.time();print(search.binary_search(L,k));b=time.time();print(b-a)
0
0.00022196769714355547

In [40]:
```

So, it is the technique also apart from the fix it technology. You cannot better. In fact, you use a super-duper computer to compute this using this algorithm, and you use a very old computer from the 80s to compute to which runs binary search, this will still beat your modern computer, because the technique that you are using matters. Alright, long story short, your algorithm matters.

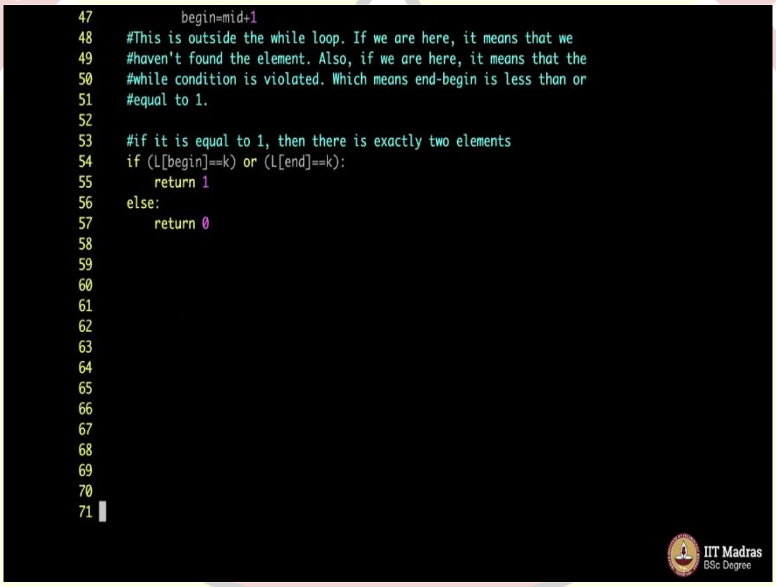
(Refer Slide Time: 45:38)



```

def obvious_search(L,k):
    '''Check if a given element k
    is present in a list L or not. This function
    was authored by S. R. S. Iyengar'''
    for x in L:
        if x==k:
            return 1
    return 0
    #code verified. Working fine.

'''A question: Can we write a piece of code that
searches for a given element in the list L faster than the obvious algorithm
given above :-( :-( :-('''

def binary_search(L,k):
    '''This function is an alternative for the obvious search. It
    does exactly what is expected from the obvious_search, but in
    an efficient way. This method is popularly called the
    binary search.'''

    #We want to shrink my list
    #We will do that using a while loop.

    begin=0 #first element in L. L[0]
```



```
begin=0 #first element in L. L[0]
end=len(L)-1 #the last element in L is in len(L). L[len(L)-1]

#Use a while loop to look at the list and keep halving it.
while(end-begin>1):
    #we will handle the case when the number of elements is less than or
    #equal to 1

    #Compute the mid which is the mid point of begin to end.
    mid=(begin+end)//2
    #if mid is indeed k, then we return True and stop the code.
    if (L[mid]==k):
        return 1

    #if the middle element is greater than k, then cut the right side and
    #retain the left side.
    if (L[mid]>k):
        end=mid-1

    #if the middle element is less than k, then cut the left side and
    #retain the right side.
    if (L[mid]<k):
        begin=mid+1
#This is outside the while loop. If we are here, it means that we
```



```
        begin=mid+1
#This is outside the while loop. If we are here, it means that we
#haven't found the element. Also, if we are here, it means that the
#while condition is violated. Which means end-begin is less than or
#equal to 1.

#if it is equal to 1, then there is exactly two elements
if (L[begin]==k) or (L[end]==k):
    return 1
else:
    return 0
```

What was the algorithm let me, let us take a look at the algorithm once, because it has worked like magic, although, slightly big here binary search tried reducing your space of sum and all the elements L. The region of interest shrunk by half and half and half and very quickly you could see, whether k belong there or not the code was so very simple.

Try doing it. Anyways I hope, you all enjoyed this piece of code. First time, we wrote some non-trivial code quite long a code and then quite deep in logic. Maybe as and always it is not easy to get something in the first run. You may have to go through it again and again. So, try coding with me a couple of times, redo the video. I mean, rerun the video and redo the code and it will be very clear to you. If it is not clear to you, as and always we are there to help in live session or on the phone. Thank you.