Python Data Structures:

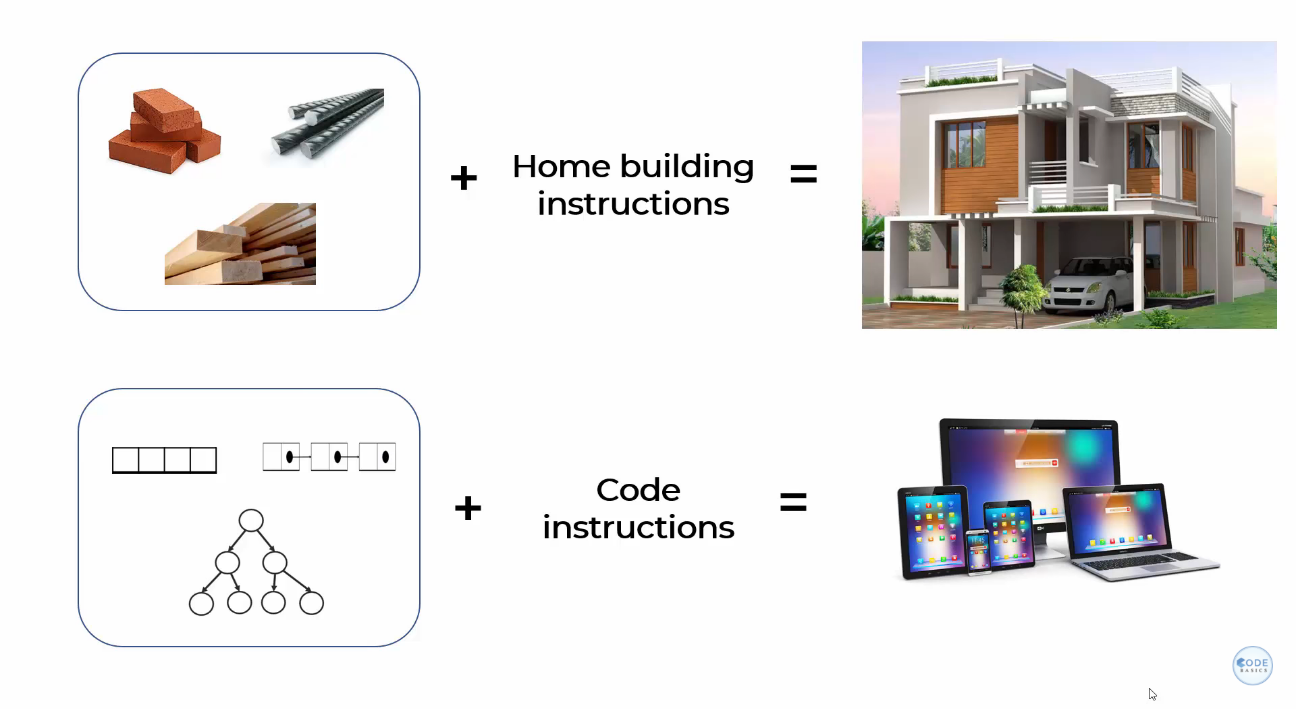
I haven’t heard about any programming interview where they don’t ask you about data structure

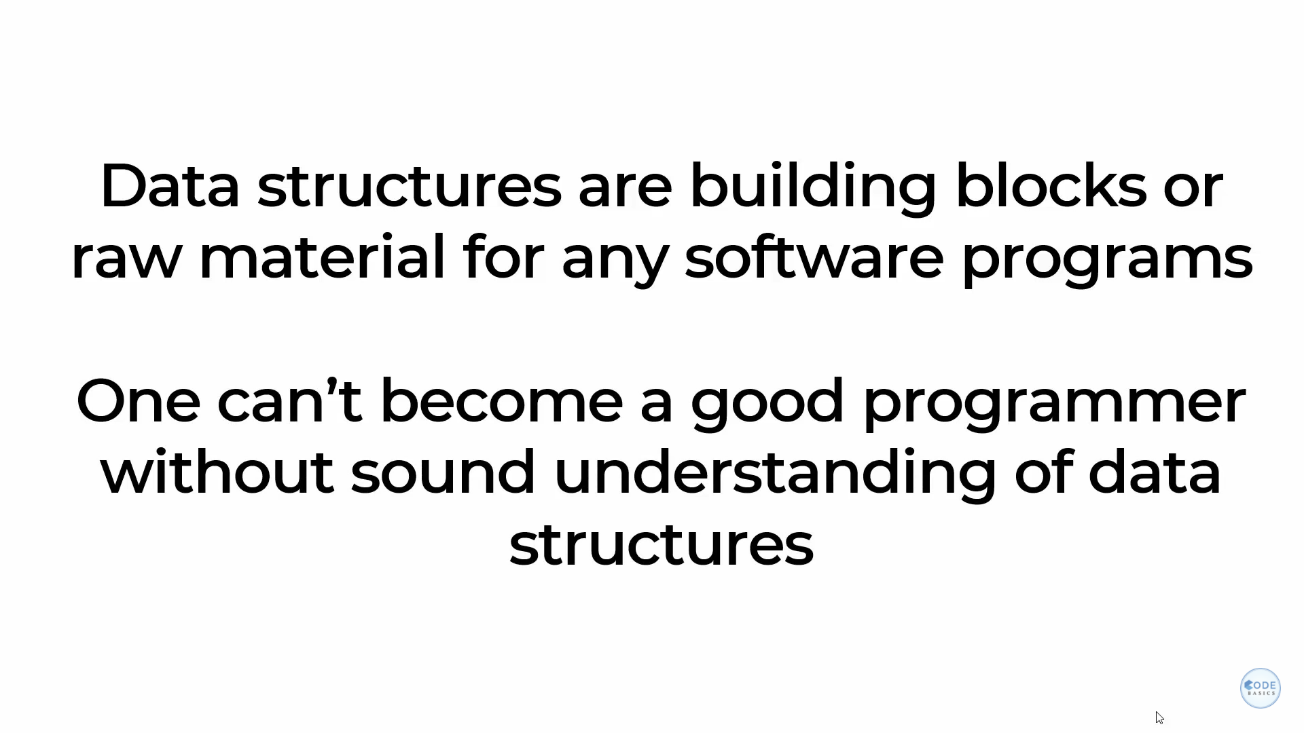
What is data structure and why are they so important?

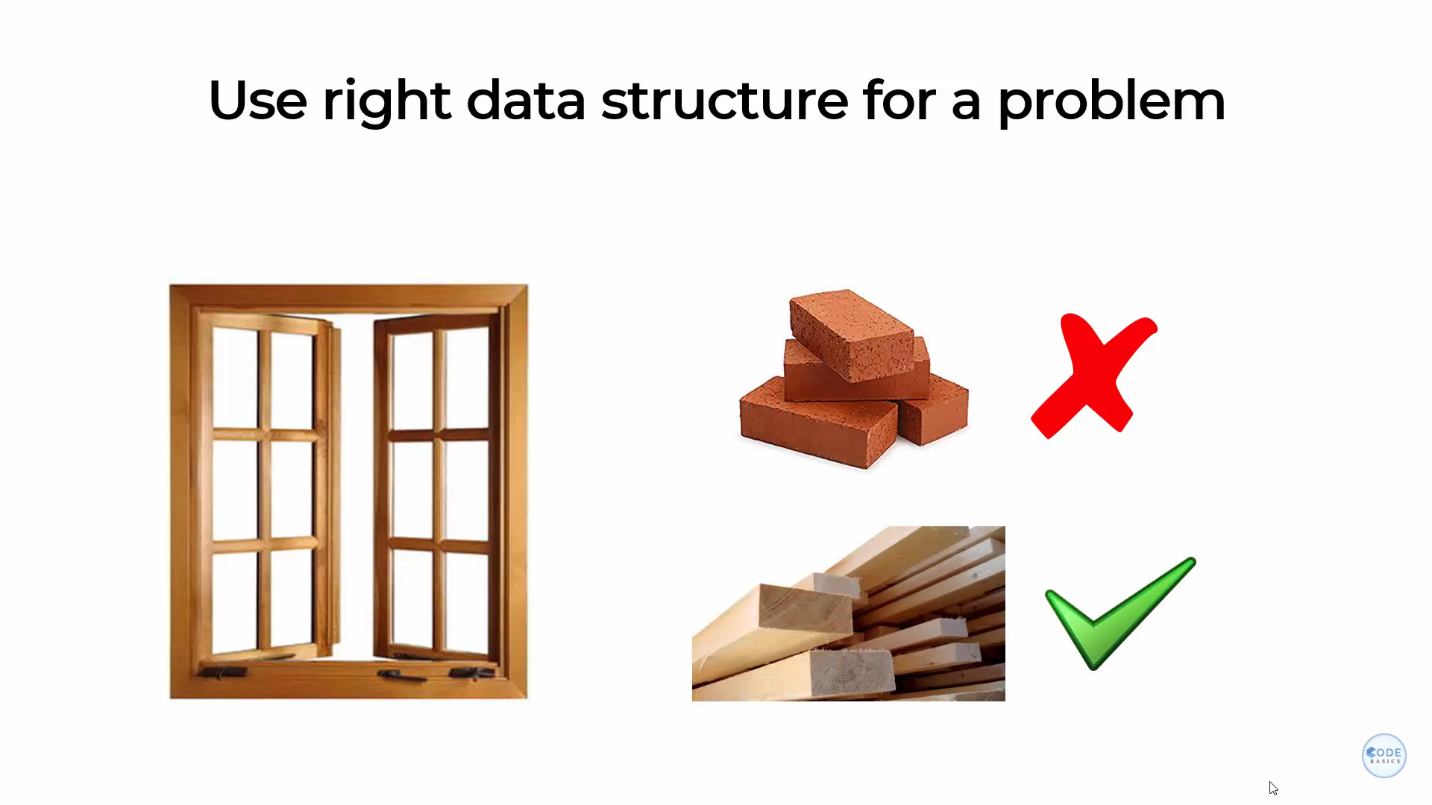
When you building a home, you need certain building blocks and raw materials such as bricks, wooden planks to build your furniture, metal rods etc.

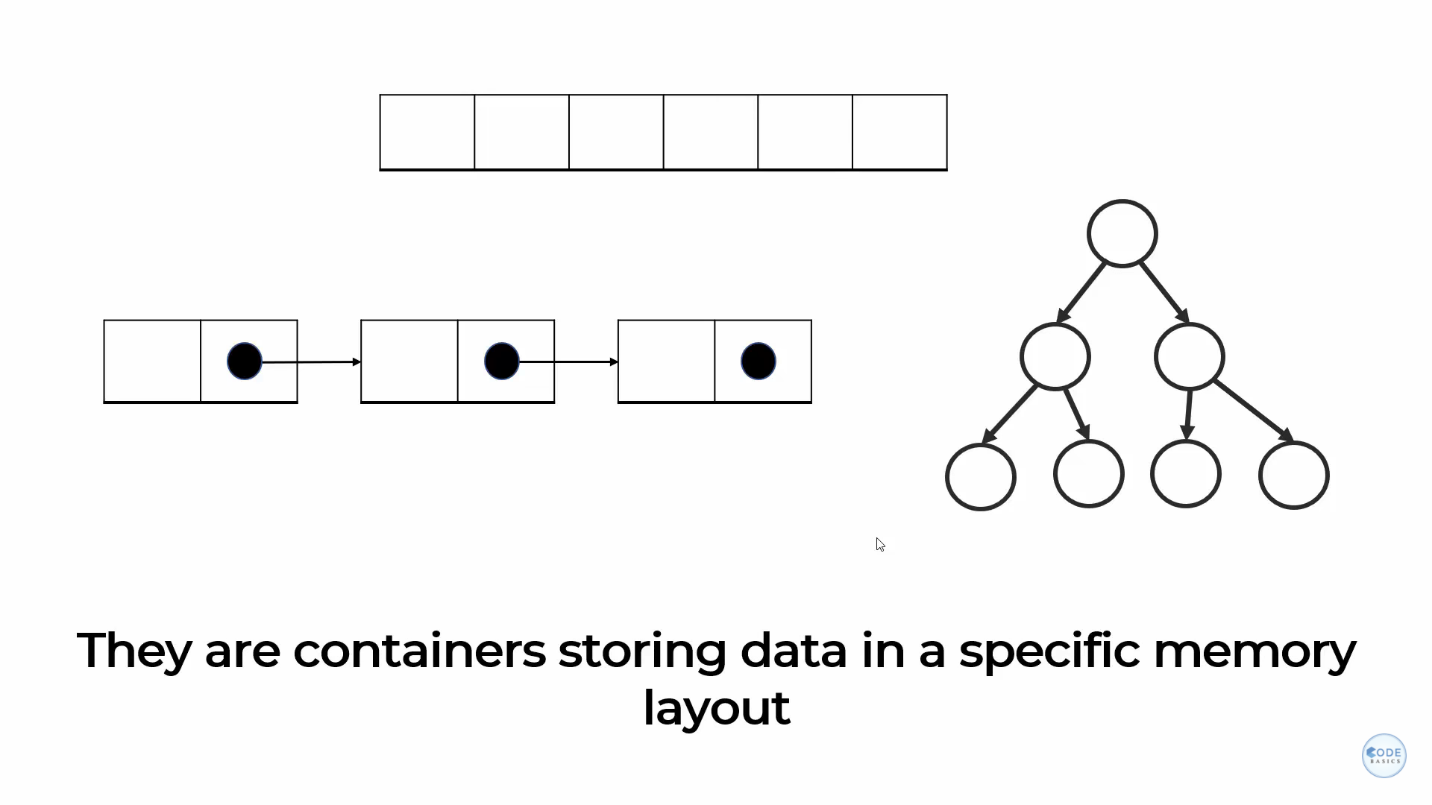
On top of that you apply the instructions for building a home and you get a home as a result.

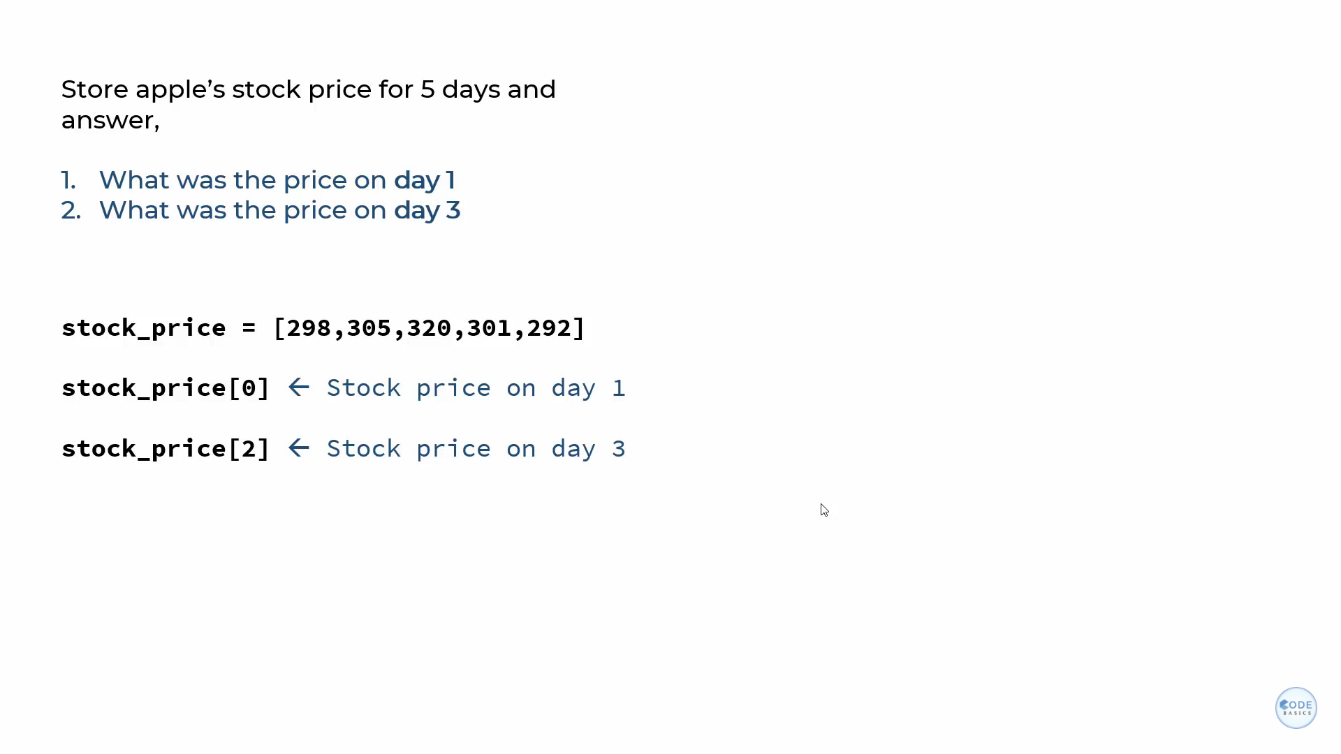
Similarly, when you’re building a software application you need some raw building blocks which are basically your data structure such as array, linked list, tree etc. On top of that, you apply the code instructions which operates on those data structures and as a result you get a software application.





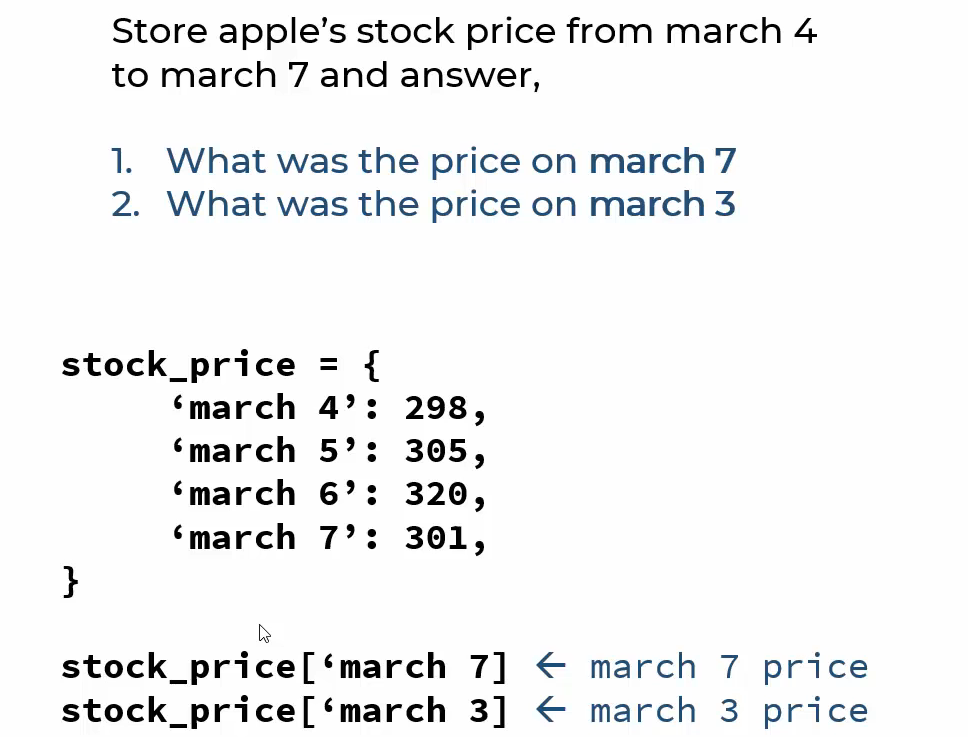
And the reason is you need to 

As a software engineer your job is to pick the right data structure for a right problem. Here I have a visual representation of certain data structures such as at the top I have array, then linked list and then tree.

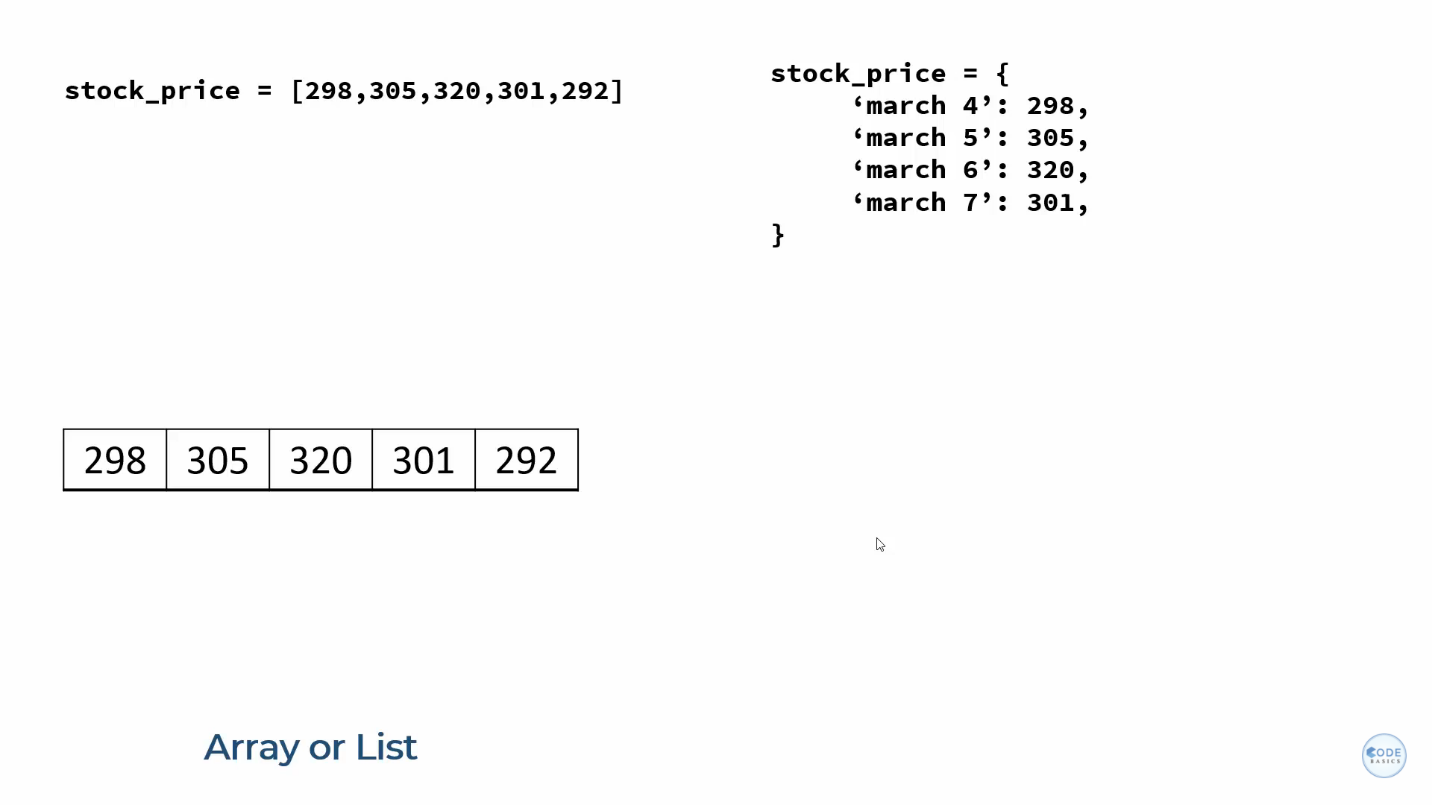
For example, you want to represent apple’s stock price for 5 days and you want to ask questions to your program such as what was my price on day 1? and what was my price on day 3? So here you only care about sequence, you’re caring about day 1 vs day2 vs day 3 and so on. In this case you will write a simple python program which look like this: -

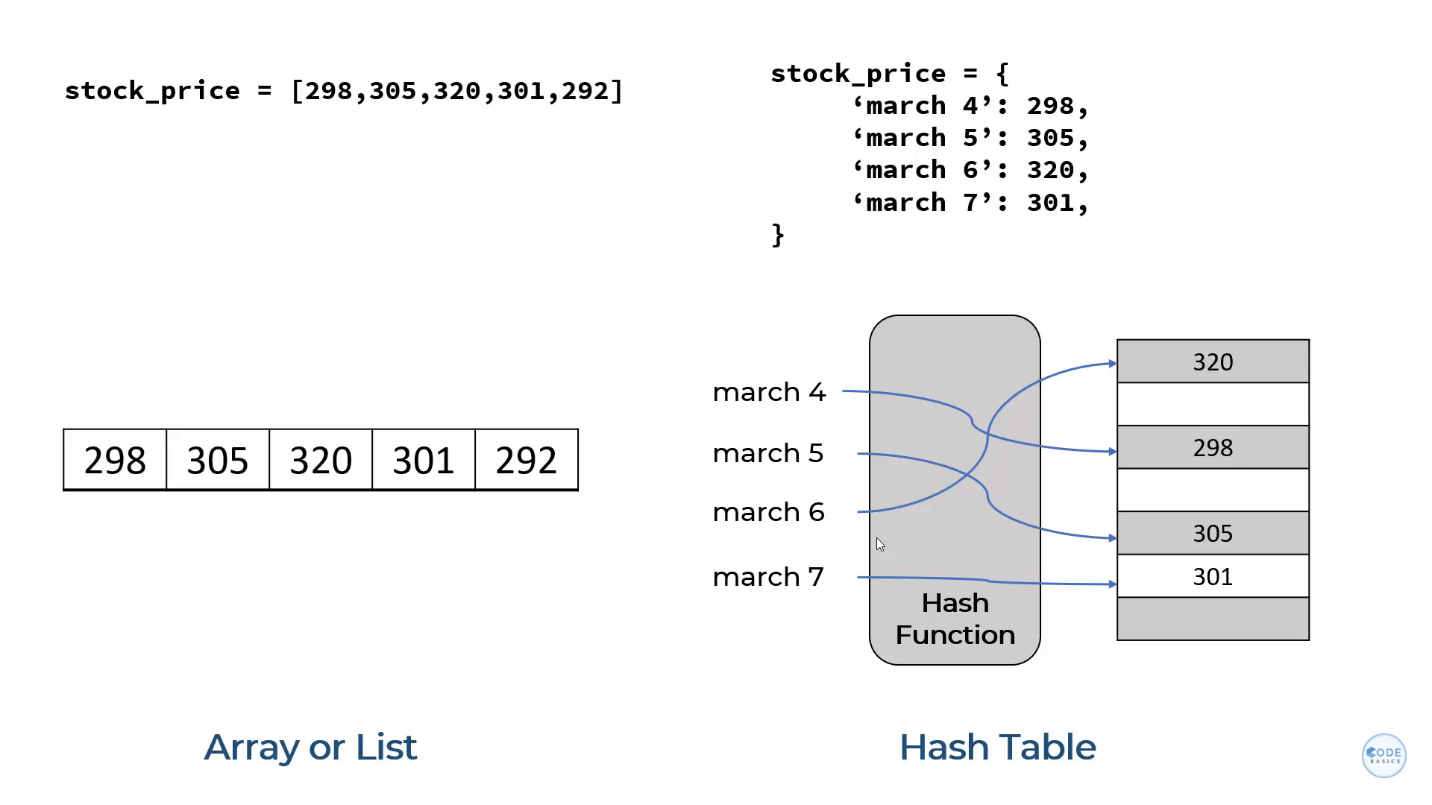
Which stores the stock prices in a list, and you can access that list by using an index operator. And you can get the price on a certain day.

But let’s say the problem is something like you want to store stock prices along with the dates. And the question you are asking to the software program is what was the price on the given date IN this case it make sense to use a dictionary. Where you can access the specific price on a day by using a dictionary key.

So, you can see that I used two different data structures which was a list and dictionary which is actually hash map behind the scene I used these two data structures to solve two different problems. So, it depends on what kind of problem you’re solving based on that you need to pick the right data structure.

Now if you look at the memory layout of both of these data structures for an array it looks something like below where the nos. are stored in a contiguous memory location and when I say memory I’m referring to a RAM.

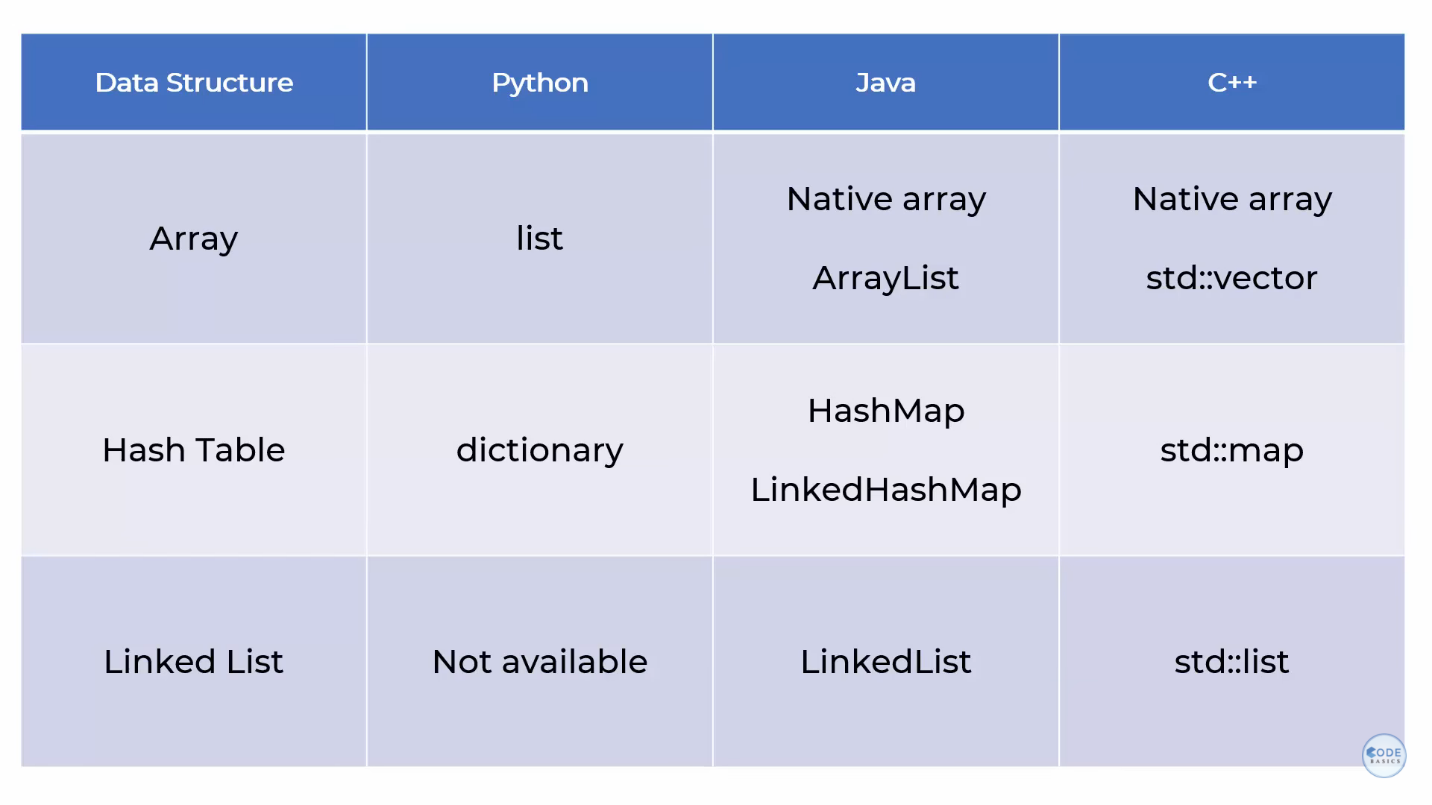


In case of dictionary, it uses hash maps as a data structure behind the scene and the way hash mapworks is you have a key on which you apply hash function which will give you an address of a bucket and using that address now you access a given element. This way you get order of one complexity for your search and your search is really fast. 

Now if you are not good in Data Structures you might end up using array for problem number 2 above which is on right hand side and it will work but the program will not be as efficient as it will be hard to read and it will incur a technical death.

Hence Programming is art as well as science and the art skills you will develop as a programmer as you gain more in depth understanding of data structures.

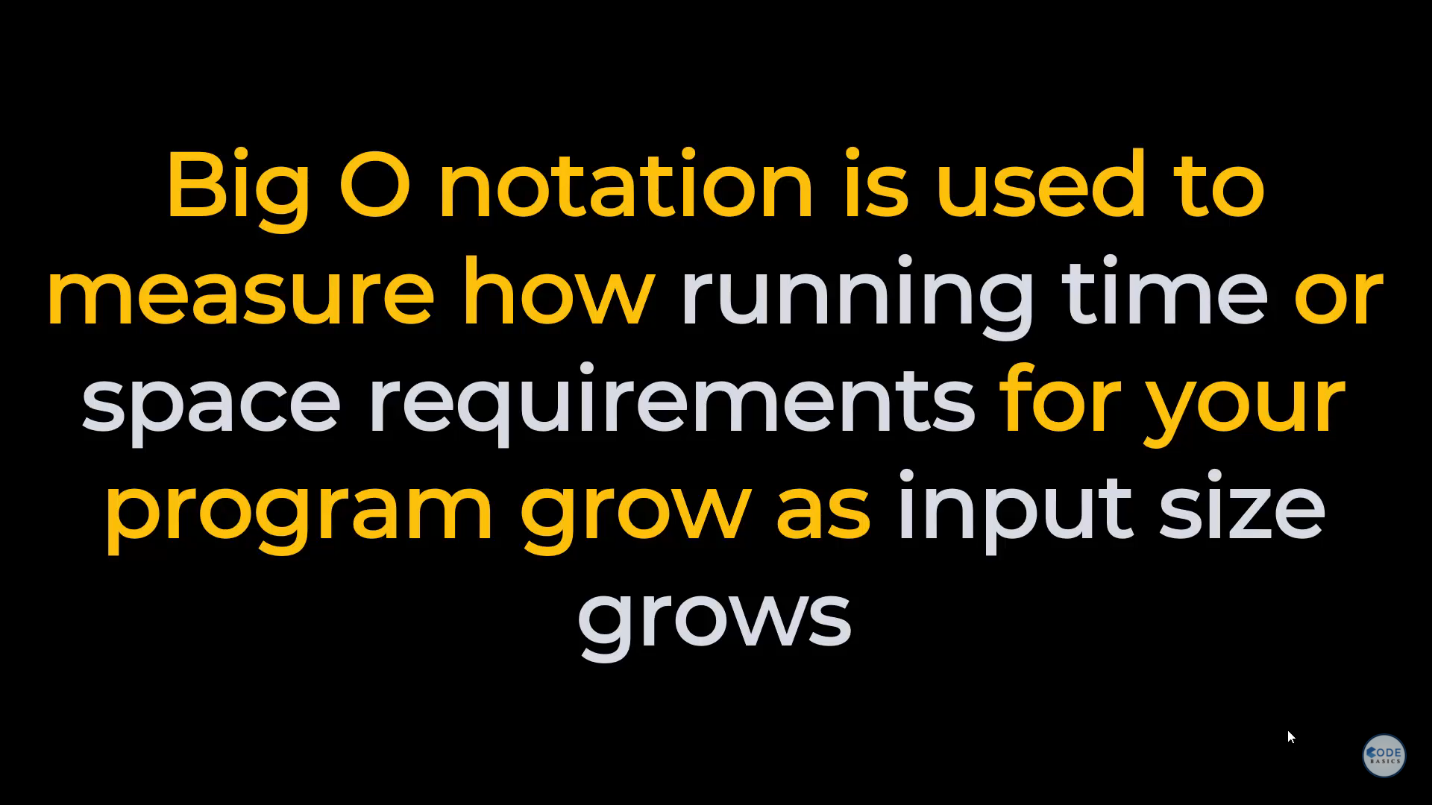
So, if you are becoming a data analyst, scientist or a programmer it is very important that you have a good understanding of data structures.

Here I have outlined some common data structures implementation in different languages. 

So, remember that data structures are a conceptual thing, different programming languages implements data structures in different ways for example: - python has a list as an implementation of array data structure, in java arrays are implemented in two different ways one is native array which is static array and array list which is a dynamic array. Will talk about it later.

The first thing that you have to make yourself familiar with when you start learning data structure and algorithms is BIG O notation.

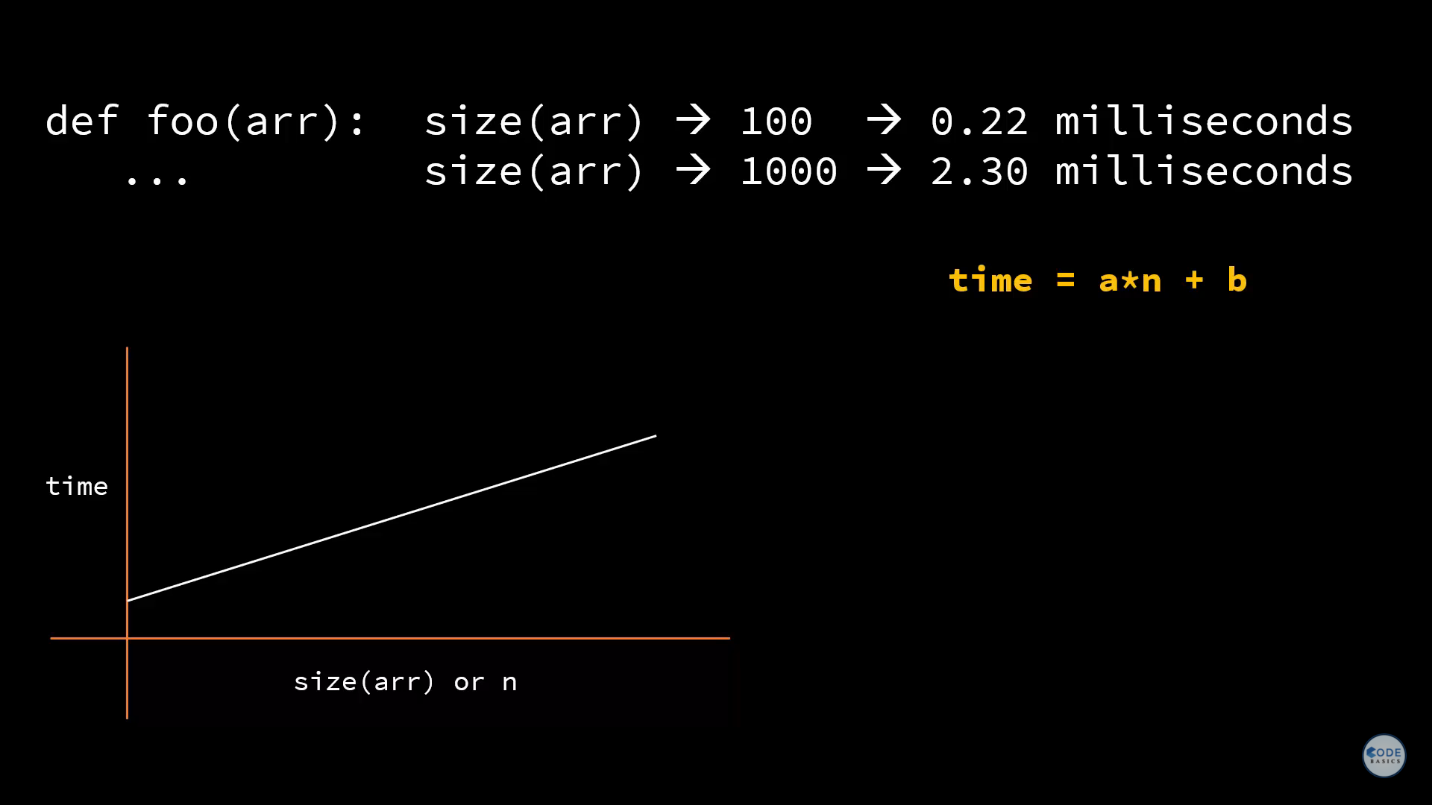
I haven’t seen any programming interview where they don’t ask about Big O notation and data structures and algorithms.



Here we are only talking about running time.

Let’s say you have a function which takes array or a list as Input and it does some processing and it returns some output.

When the size of array is 100 it runs in 0.22 milliseconds and when size of array is 1000 it takes 2.30 milliseconds.



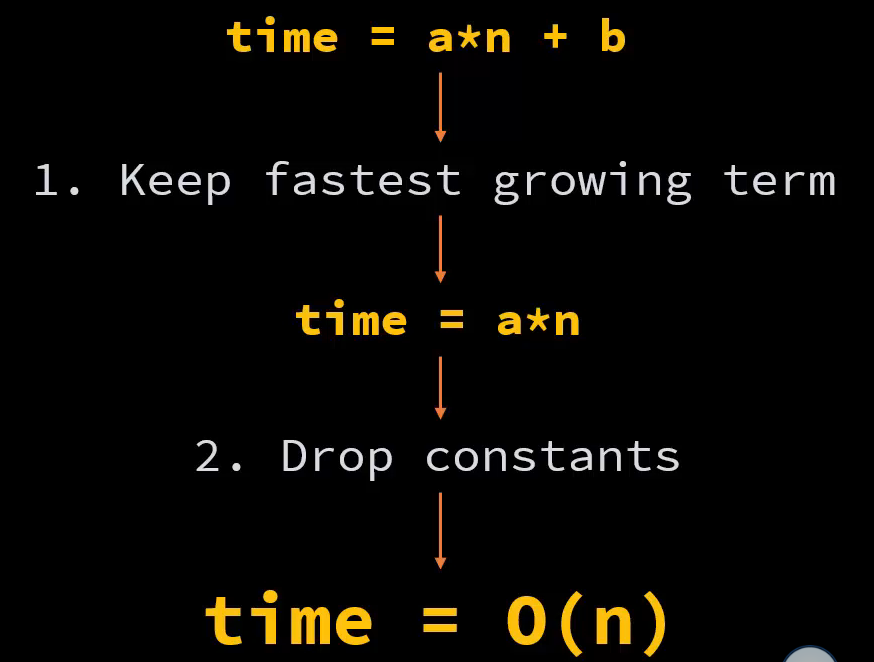
So, as the size grows the time also grows and let’s above attached is the graph of size vs time you can clearly see that it’s a linear function. The time that it takes to run the program is linearly proportional to the array size and any linear function can be represented as (*a \* n) + b*. Now the reason we are using a mathematical function here to measure the time requirement is because depending upon the computer this time might have different. I have a fast computer so let’s say it’s 2.30 millisecond but if you’re on a slow computer it might going to take 5 milliseconds. So, measuring time of program in absolute terms is not useful. It has to be mathematically representant and that mathematical representation is Big O.

Here we are not going to say the time is Big O((*a \* n) + b), but we* are going to apply some rules.

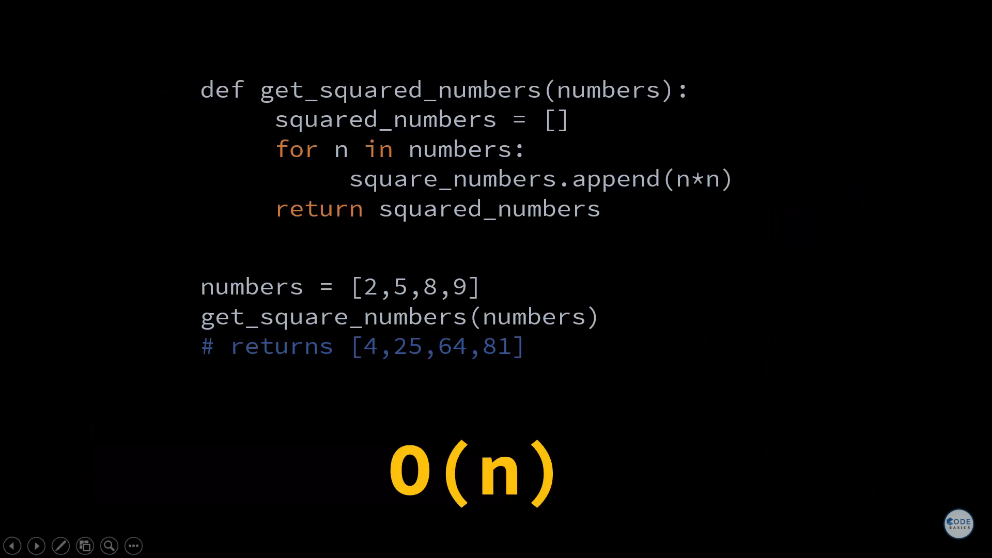
So, the first rule is –1. We are only going to keep the fastest growing term. The b here is constant so as the size of n grows let’s say n is 100 and a is 2 so it will be 200 but when n becomes million so it will be 2 million vs b remains constant so the fastest growing term is a \* n. So, we only keep that.

The second rule is we drop any constants. So, constant is a and we drop it so what is left? Is n.

And now we say the Big O time complexity of this program is order of n.

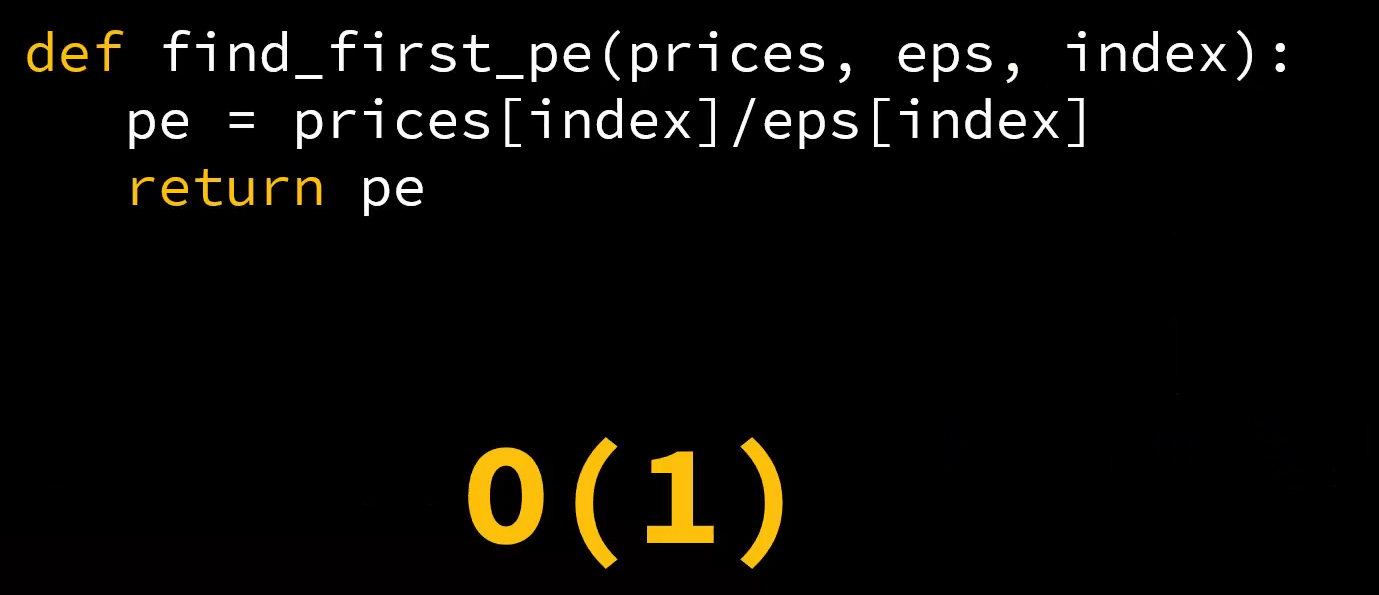


Now below is the example of a program whose time complexity is O(n) means order of n.

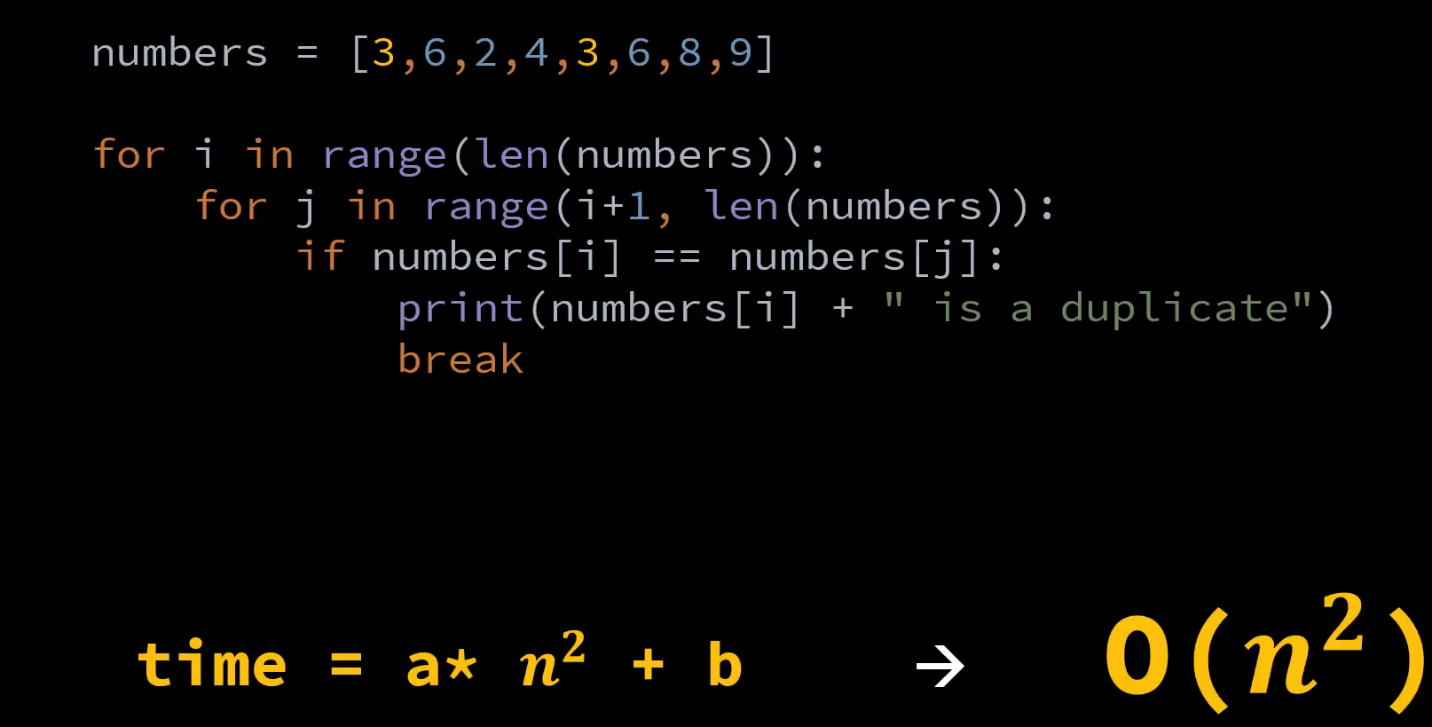


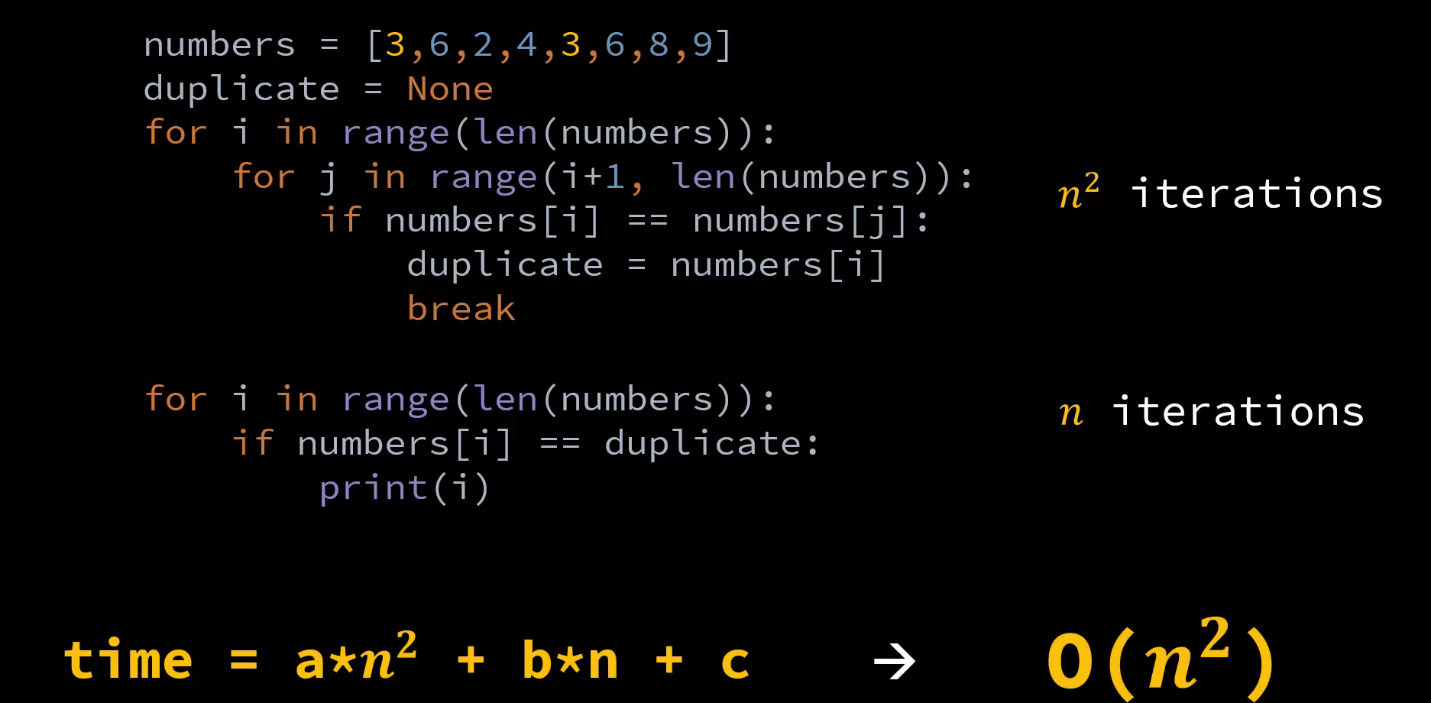
Let’s say you are squaring the numbers by taking an input list so above is the python program. Now let me clarify this that Big O notation is used for all programming languages. It’s just not specific to python. I’m just using python for example here. In this example you are running n iterations. So, right now my array size is 4 so it did 4 iterations but let’s say my array size is 4 million then it’s goanna do 4 million iterations. And as you know the time it takes to iterate the program is equally proportional to number of computations it is doing. Here if the size is 4 million it is goanna do 4 million computations in it’s for loop. Hence the time will grow linearly therefore, the time complexity of this program is O(n).

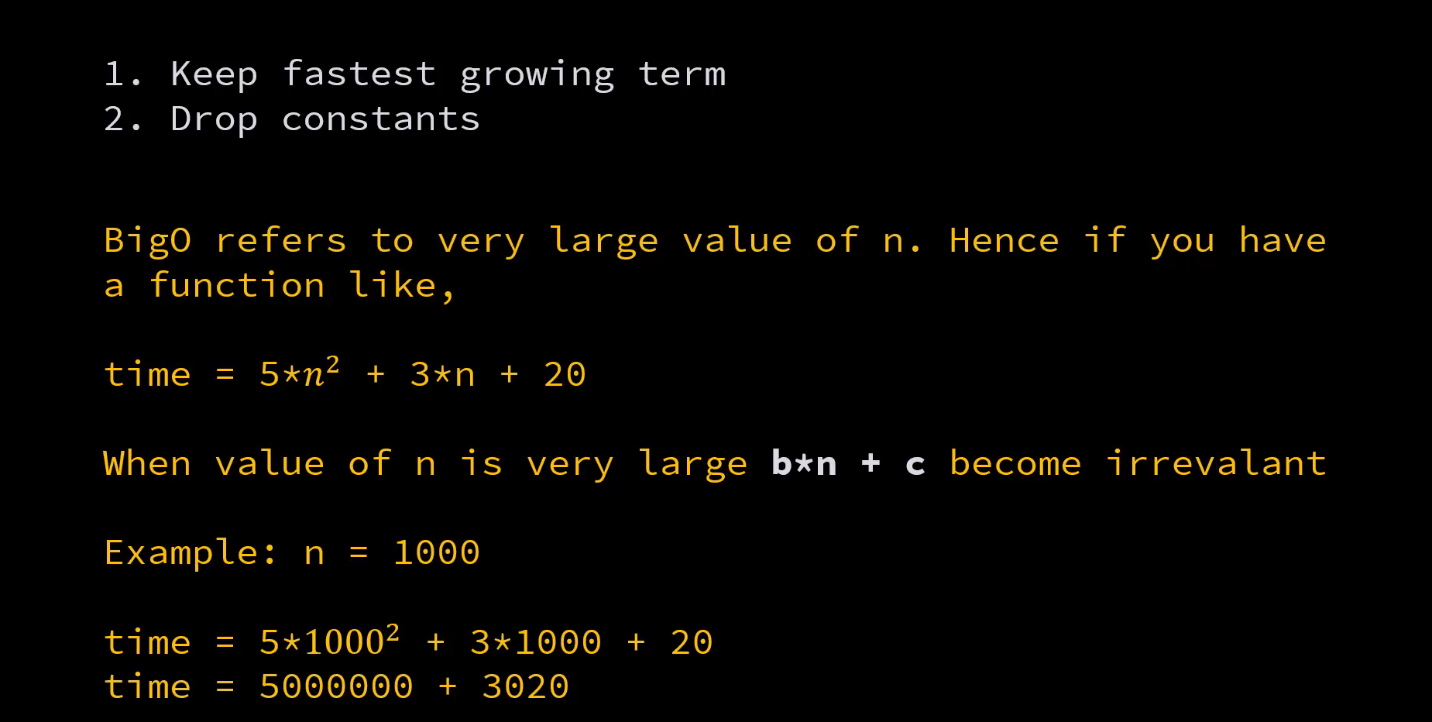
Here is another example where when you increase the size of the input the time is almost constant it changes but only little bit. So, you see like .22 and .23 milliseconds. So, the graphical representation of the time as input grows could be this as shown below. And it’s a constant function and constant function can be represented by time = a. And when you apply those two rules what you get is order of one complexity O(1). Because the way you look at a is a \* 1 and by the second row you drop that constant which is a. And what you get is order of 1.

Here is an example of a function whose order of complexity is 1. Let’s say you are trying to find the pe of 1st stock in your list. So, here I’m getting the stock price in a list, I’m getting eps of different stocks in the list. So, these two lists have price and eps for different stocks there could be 10 stocks, million stocks but when you supply an index it is going to do a constant operation. So, it doesn’t matter the size of price or eps it could be 10 or 10 million the time execution is going to remain constant. Hence, the Big O complexity of this program is order of 1.

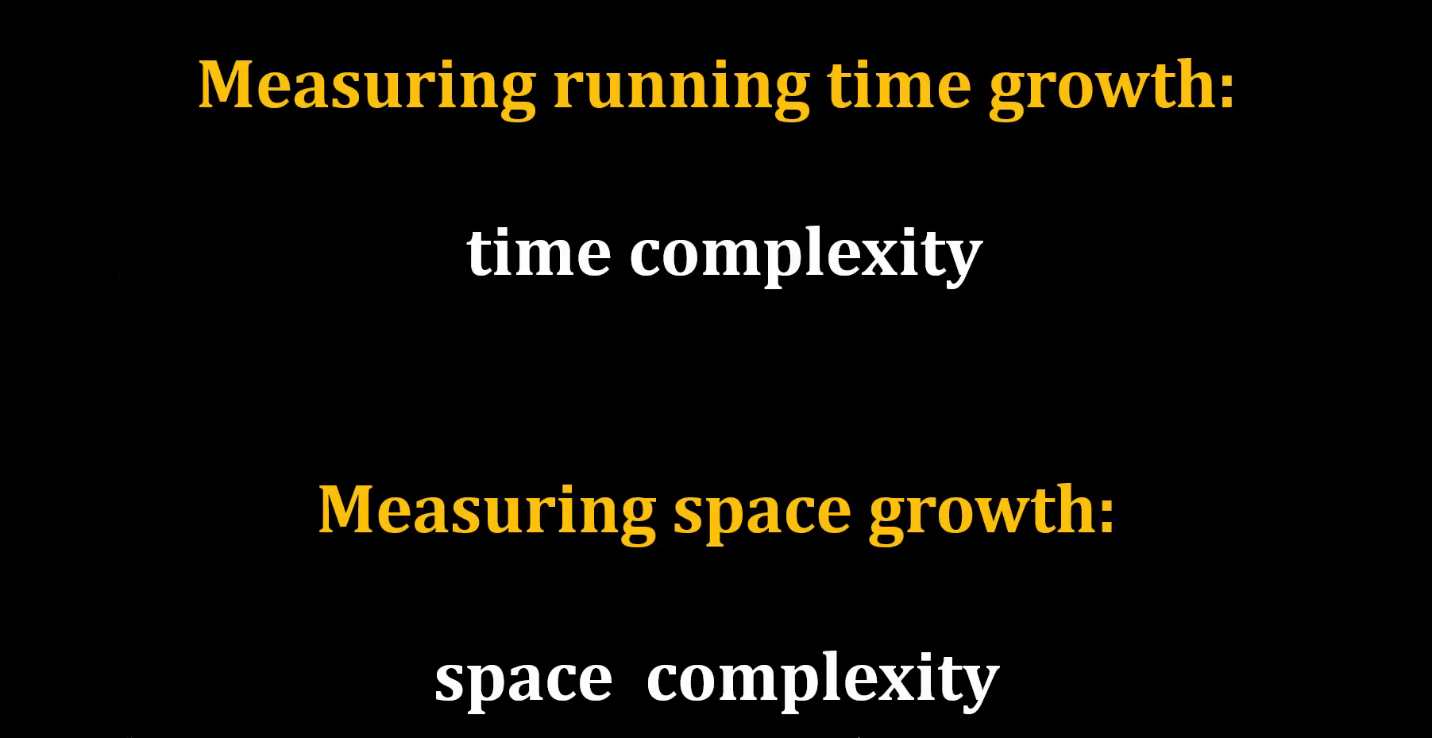
Below is another program where you are trying to find a duplicate from your list. Here we’re running two for loops and you are comparing the numbers in two for loops and trying to find the duplicate. You might be aware about this program. The linear equation for the timing of this function as the list grows is

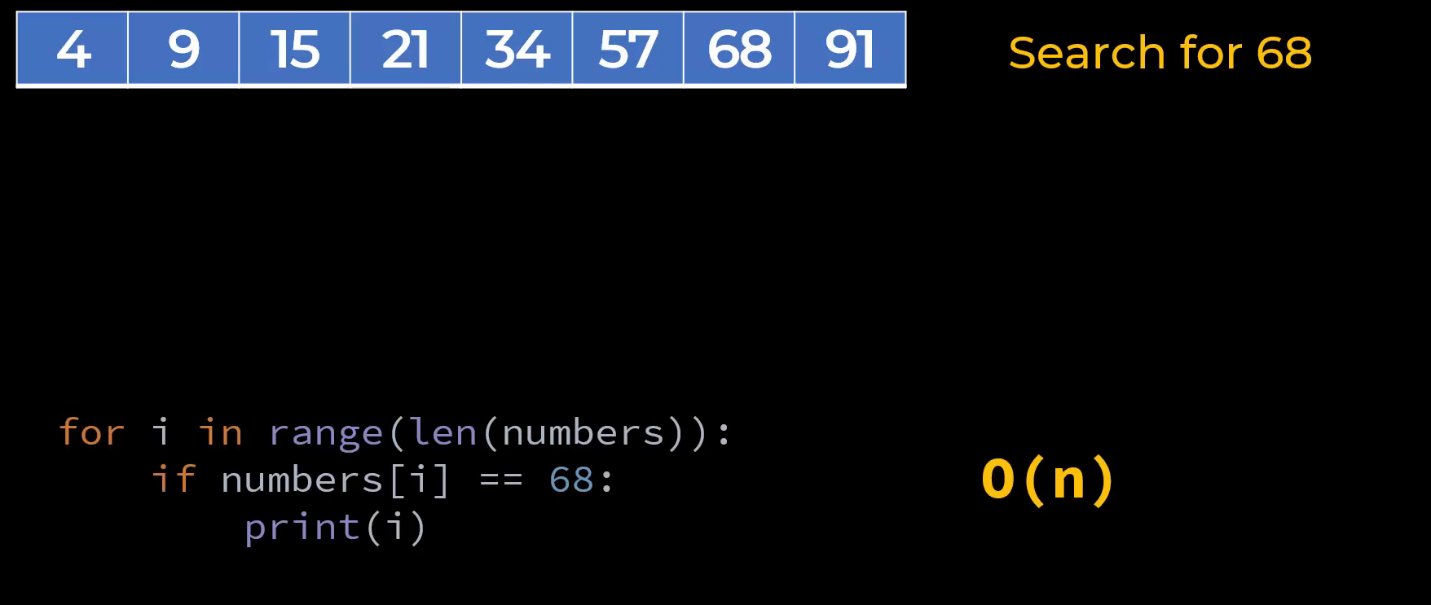
a \* n^2 + b. This is the standard linear equation okay. And here if you give only the fastest growing term which is a \* n^2 and drop the constant such as b and a you will get order of O(n^2).

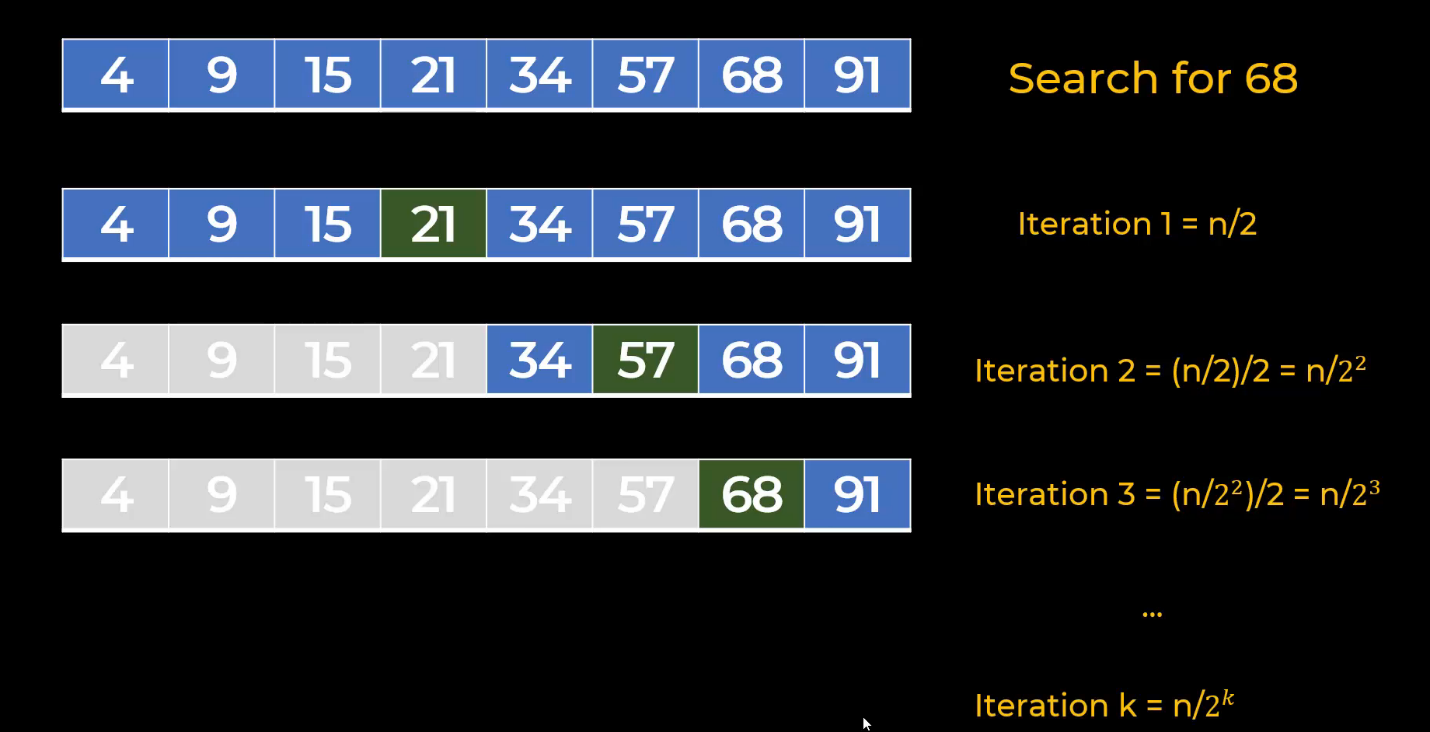
Here is another example where in this program the first block of code is doing n^2 iterations and second block of code is doing n iterations. So, to represent the time you can use🡪 time = a\*n^2 + b\* n + c. But again, in this equation when we apply those two rules keep the fastest growing term which is a\*n^2 drop b and c and then also drop the constant which is a and then you get the order of O(n^2). So, sometime people say the complexity of this program is O(n^2) + n but no it is order of n^2.

And I’ll explain you why you apply those two rules, why you keep only the fastest growing term. Let’s understand that the reason behind those two rules. Let’s say you have a function which is something like time = 5\*n^2 + 3 \*n + 20. Now Big O is very rough ball park estimate it refers to the value of n which is very large. I’m not going for any large value here but let’s say even if n value is 1000 in that case in this equation the first term will contribute the most it will be 5000000 whereas the 2nd and 3rd term will come around 3020. So, 3020 is very minor in 5 million value. Hence, it can be dropped.

Big O is a simplistic method of saying this is how much my program is going to take. Hence, it makes sense that you only keep the fastest growing term and drop everything else.

Until now whatever we saw was for running time growth it is called time complexity and the same can be used to measure the space growth and that is called space complexity.

Let’s look at one more example of binary search, when you have a sorted list of numbers. If you want to search for a specific number, how do you do that? In this example I have 8 numbers and I want to search for 68. Simple method is you go through numbers one by one and compare against 68 and find the index. This has order of n complexity.

This is a very simple example so this program will run very fast but let’s say you have billion numbers in your list and you want to find out index of a specific number you don’t want to do build an iteration because it might be very time consuming. There is a better way which is binary search, in this what you do is you first find the middle element which is 21 you compare it against 68 you find out that 21 is less than 68. Hence, you now discard the array on the left-hand side. So, numbers 4-21 you discard it. And now what you have 34, 57, 68, 91. In that array also you find the middle element. So, you basically in each iteration you are dividing the array by 2. Here the middle element is 57. Again, compare it to 68, 57 is less than 68. Hence, you discard the left-hand side array which is 34 and 57. You now have only 68 and 91 in your search space. Again, you divide it by 2 you take the middle element which is 68 and this is the element you are looking for. So, you found your answer in 3 iterations. If we have done n iterations which is the previous approach, we would have taken 7 iterations because the element is at the 7th number. So, this is better. If you have big array and if you are doing like k iteration, so equation on the right hand side in image below shows that for iteration k you have to do n/2^k operation for k iteration.

Now let’s convert this into Big O. So, we have to do n/2^k iterations to get the array size to 1. 1 because that is our worst-case estimate. You might find your answer in the first iteration itself if the element you are looking for is exactly in the middle.

But Big O notation is used generally to measure performance