

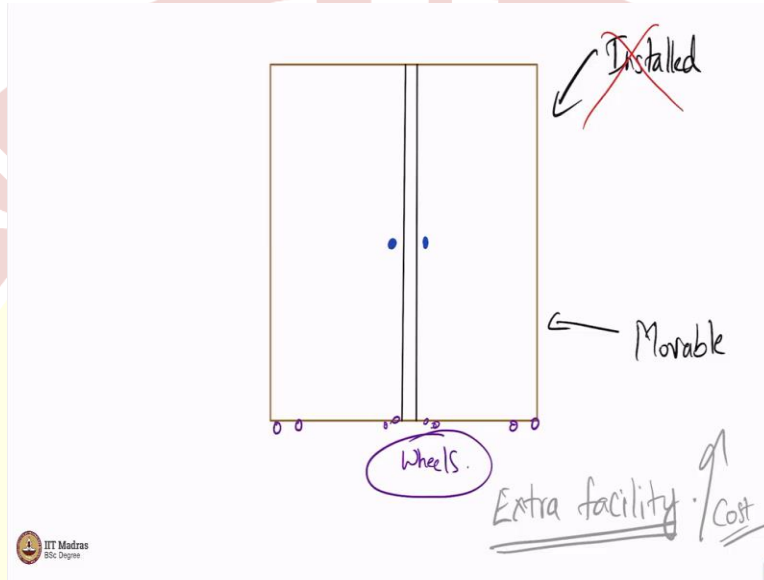


IIT Madras

ONLINE DEGREE

Programming in Python
Professor. Sudarshan Iyengar
Department of Computer Science & Engineering
Indian Institute of Technology, Ropar
Tuples

(Refer Slide Time: 00:16)



So, let us consider a small example. As you might have this cupboard, it is a cupboard, so how does it look like. Let me try writing it with my poor drawing skills. So, you see this is the cupboard I can think of. Thanks to the modern drawing technology. Very bad rectangle or a square becomes proper. So, people like us who are not very good at sketching can still manage. So, how does a keyboard look like? So, let me put a vertical line here. Then again, thanks to the drawing technology. It makes it vertical, very good. So far, so good. Maybe you should put this exactly to the center. You see it even places it in the center.

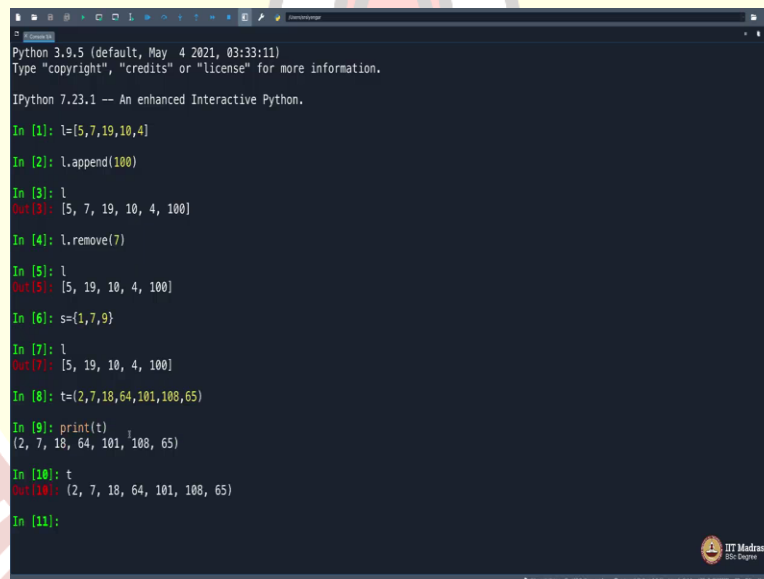
So, still does not look like a cupboard, an almirah or whatever, a closet depends on which part of the world you are in so maybe a handle here and a handle here. And you can open this cupboard. So, now, imagine this cupboard is installed in a particular place. You do not change the place, you see. You keep it like that. It is not, it is kept in a permanent place in your house. You do not change it or shift it.

Let us say as opposed to this, I want a cupboard which keeps moving. For some reason I keep moving it here and there. It is not installed in a particular place, but moved around. I want that to

be mobile, let us say. I want this to be movable and mobile. Then what would I do, I would rather affix some wheels to it, you see. I will put some wheels to it. Some sort of wheel so that it is easier to carry it around as simple as that. There are some wheels here. And the cost is slightly more because you want an extra facility. This becomes an extra facility, you see. This is a little extra facility, that is it. That is more observation. You must be wondering what are we doing.

Sometimes when a teacher is not well prepared, he starts talking obvious things. So, maybe yes, I am doing that. Let us see if this comes in use somewhere in my teaching of this concept called tuples in Python. So, I will finish my teaching. And at the end, you should guess, what is this to do whatever I said right now, the mobile, the not mobile facility, something that becomes extra and might even hype the cost a little. What is this to do with Python programming, and especially, the topic that we are going to discuss right now? Just watch out.

(Refer Slide Time: 3:34)



```
Python 3.9.5 (default, May 4 2021, 03:33:11)
Type "copyright", "credits" or "license()" for more information.

IPython 7.23.1 -- An enhanced Interactive Python.

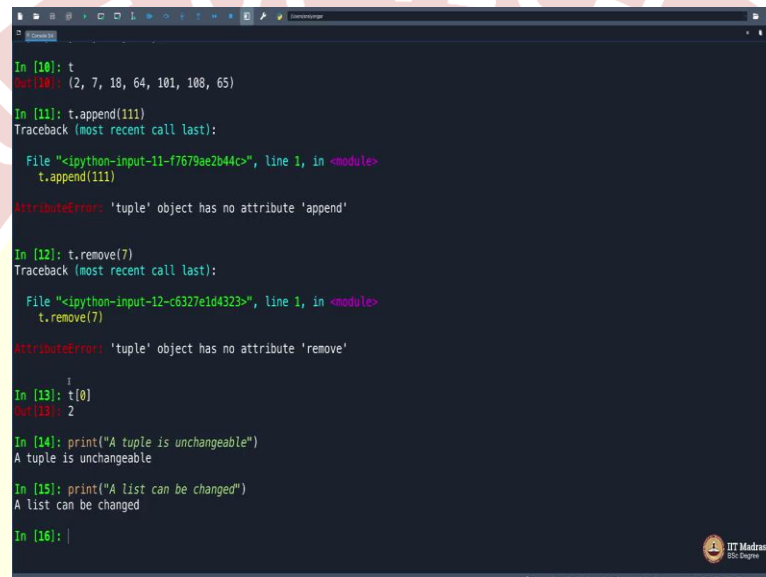
In [1]: l=[5,7,19,10,4]
In [2]: l.append(100)
In [3]: l
Out[3]: [5, 7, 19, 10, 4, 100]
In [4]: l.remove(7)
In [5]: l
Out[5]: [5, 19, 10, 4, 100]
In [6]: s={1,7,9}
In [7]: l
Out[7]: [5, 19, 10, 4, 100]
In [8]: t=(2,7,18,64,101,108,65)
In [9]: print(t)
(2, 7, 18, 64, 101, 108, 65)
In [10]: t
Out[10]: (2, 7, 18, 64, 101, 108, 65)
In [11]:
```

So, consider a list `l` is equal to submit it is here, let us say 5, 7, 19, 10. We have been doing this from a long time. In fact, we can take random numbers, but I enjoy simply typing these numbers randomly. So, you can always add something to it. You can remove something from it. Once you add you see what you have here is one extra element 100. And you can remove, let us say, an element 7 here and it gets removed, so on and so forth. This is a known fact, nothing surprising.

But then instead of doing it like this, assuming you created a variable `t` is equal to, you see for a set you used the flower bracket. For a list, you used big brackets. This is how we were taught in

high school. At least that is how I was taught in my high school. So, for set to use flower brackets, so what is left out if you think for a minute. It is the small bracket. And in fact, Python uses it for something. Let me just introduce you people to what is called tuples. So, t for tuples, t equals, if you use the small brackets, let me try that 2, 7, 18, 64, 101, 108, and let us say 65 and so on, so if you print t or simply say t, whatever, you will get this with the small brackets.

(Refer Slide Time: 5:09)

A screenshot of a Jupyter Notebook interface. The background features a large, semi-transparent watermark of the Indian Institute of Technology Madras logo. The notebook contains several code cells. The first cell creates a tuple 't' with values (2, 7, 18, 64, 101, 108, 65). The second cell attempts to call 't.append(111)', resulting in an 'AttributeError: \'tuple\' object has no attribute \'append\'' error. The third cell attempts to call 't.remove(7)', also resulting in an 'AttributeError: \'tuple\' object has no attribute \'remove\'' error. The fourth cell prints 't[0]', which outputs '2'. The fifth cell prints the string 'A tuple is unchangeable'. The sixth cell prints the string 'A list can be changed'. The seventh cell is empty.

```
In [10]: t
Out[10]: (2, 7, 18, 64, 101, 108, 65)

In [11]: t.append(111)
Traceback (most recent call last):
  File "<ipython-input-11-f7679ae2b44c>", line 1, in <module>
    t.append(111)
AttributeError: 'tuple' object has no attribute 'append'

In [12]: t.remove(7)
Traceback (most recent call last):
  File "<ipython-input-12-c6327e1d4323>", line 1, in <module>
    t.remove(7)
AttributeError: 'tuple' object has no attribute 'remove'

In [13]: t[0]
Out[13]: 2

In [14]: print("A tuple is unchangeable")
A tuple is unchangeable

In [15]: print("A list can be changed")
A list can be changed

In [16]:
```

But let us see something, you could do append for the list. In fact, you have so many things that you can do with lists. I would strongly suggest that you look at all of them. You can append. You can clear the entire list, copy the list on to something else, count the number of occurrences of an element, find the index of something, extend the list on to a bigger list by adding another list on to it, rather appending some list on to it, so on and so forth. There are lot of things that you can do here.

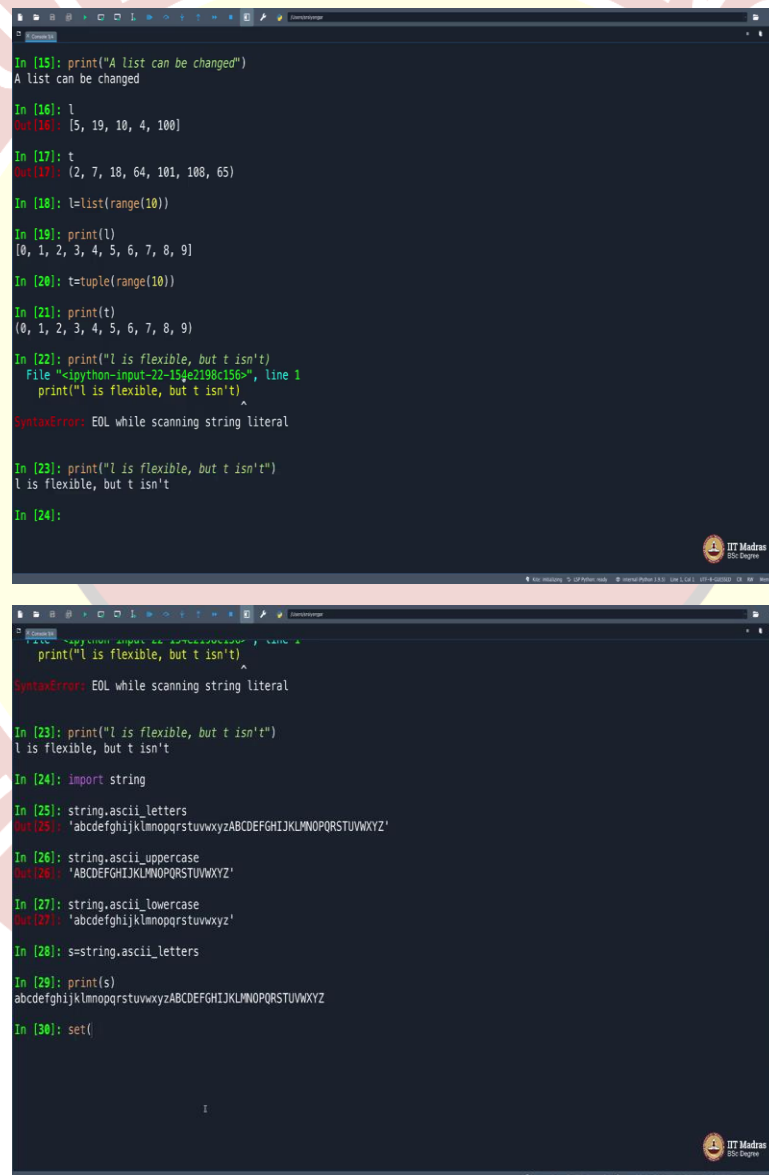
Roughly, how many do you have some, if I can complete some 11 possibilities with l. But then, if you see t, you only have two possibilities, which means the moment you create it, you cannot append anything to it. Let us try appending some 111, let us say, see it throws up an error. Let me try removing something from it. If we remove, let us say 7, which is the second element, it throws up an error. So, what is t of 0? T of 0 is 0.

You can access anything the way you access a list, but you cannot append anything, you cannot remove anything, the tuple is unchangeable. Let me print that statement so that you all observe it.

A tuple is unchangeable, whereas a list can be changed. So, this is, there is a fancy word for this. Immutable and mutable copied from the word mutation in biology, but do not break your head. The idea here is one can be changed, other cannot be changed.

You can ask me why? Why would anyone want to have tuples this way? How easy would it be if we were to change tuples, but then what would be the difference between a list and a tuple if you can change the tuple to, except for the way you write the brackets, everything would be the same.

(Refer Slide Time: 7:12)



```
In [15]: print("A list can be changed")
A list can be changed

In [16]: l
Out[16]: [5, 19, 10, 4, 100]

In [17]: t
Out[17]: (2, 7, 18, 64, 101, 108, 65)

In [18]: l=list(range(10))

In [19]: print(l)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [20]: t=tuple(range(10))

In [21]: print(t)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

In [22]: print("l is flexible, but t isn't")
File "c:\python-input-22-154e2198c156", line 1
print("l is flexible, but t isn't")
^
SyntaxError: EOL while scanning string literal

In [23]: print("l is flexible, but t isn't")
l is flexible, but t isn't

In [24]:
```

```
In [23]: print("l is flexible, but t isn't")
l is flexible, but t isn't

In [24]: import string

In [25]: string.ascii_letters
Out[25]: 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

In [26]: string.ascii_uppercase
Out[26]: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

In [27]: string.ascii_lowercase
Out[27]: 'abcdefghijklmnopqrstuvwxyz'

In [28]: s=string.ascii_letters

In [29]: print(s)
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

In [30]: set()
```

So, let me think then what could be the difference between a list and a tuple? So, let me do one thing. Let me create a list of 10 numbers only, the same list of 10 numbers, which is, if you print

we will see 0 to 9 and then I will say tuple to, t equals you would have guessed it right, you will say tuple range of 10 print t or simply t. You will see this. The difference between l and t is that l can be changed, t cannot be changed. So, print l is flexible, but t is not.

You are even wondering why is he printing these statements. This is just to sort of, I mean, imprint this on your mind so that, it is like using a board, let us say, blackboard. Of course, I can use my iPad to write, but then it is easier this way. So, l is flexible, but t is not. So, what is the point in having a t? This sounds like the example that you just now saw, a cupboard that has wheels and a keyboard that does not have wheels.

You cannot say this turbo does not have wheels, so I do not want it. You probably want a cupboard without wheels for some reason that you do not want the keyboard to be movable at any cost. You want it steadily kept in a corner. Whereas maybe you have a small cupboard, but you want to move around, let us say take it near your bed when you are sleeping and pull it near your living room when you are working. Maybe it has all your laptop and other digital devices. You see there is probably an application of having something or not having something.

Now, if you are wondering what is the application of a tuple structure like this as opposed to a list structure, list option in Python, tuple option in Python, what is the difference? Sometimes you may not want things to change. You may want to fix things. And you do not want that to be changed. For example, you can put, let us say, I will call it alpha and then I will create all the alphabets a, b and so on up to z.

In fact, this is smarter way to do this. I use the smart way. There is something called import string, bear with me, string dot there is a function. If I remember it right, it is here ASCII letters. ASCII letters will have all the English alphabet letters, lowercase and upper case. If you want only upper case maybe you have what is called upper case. You want only lower case, obviously, you would say lower case.

What am I up to? Why am I even using this? For me, I would say, I will take a s equals string of ASCII characters, ASCII dot underscore letters and then print s. You have everything, wonderful. So, it was very easy instead of typing, you see. I mean, although I would prefer that you type it and keep it safely. So, where are we heading towards, just observe. I will create a tuple, let us say alpha, which is, I will convert this to a set to begin with.

(Refer Slide Time: 10:38)

```
['p',
 'q',
 'r',
 's',
 't',
 'u',
 'v',
 'w',
 'x',
 'y',
 'z']

In [31]: s=set(s)

In [32]: print(s)
{'v', 'R', 'o', 'U', 'D', 's', 'C', 'y', 'g', 'n', 'V', 'X', 'l', 'd', 'H', 'r', 'f', 'a', 'i', 'Q', 'p', 'O', 'q', 'B', 'u',
 't', 'M', 'c', 'z', 'G', 'I', 'P', 'E', 'k', 'j', 'b', 'K', 'L', 'J', 'e', 'm', 'F', 'S', 'Y', 'W', 'h', 'Z', 'w', 'M', 'T',
 'x', 'A'}

In [33]: alpha=tuple(s)

In [34]: print(alpha)
('v', 'R', 'o', 'U', 'D', 's', 'C', 'y', 'g', 'n', 'V', 'X', 'l', 'd', 'H', 'r', 'f', 'a', 'i', 'Q', 'p', 'O', 'q', 'B', 'u',
 't', 'M', 'c', 'z', 'G', 'I', 'P', 'E', 'k', 'j', 'b', 'K', 'L', 'J', 'e', 'm', 'F', 'S', 'Y', 'W', 'h', 'Z', 'w', 'M', 'T',
 'x', 'A')

In [35]: s=string.ascii_letters

In [36]: alpha=tuple(list(s))

In [37]: list[s]
```

```
In [38]: print(list(s))
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
 'Y', 'Z']

In [39]: print(alpha)
('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
 'Y', 'Z')

In [40]: x='sudarshan#*&$IndiaBharath()Karnataka$Punjab#Tamil Nadu'

In [41]: l=list(x)

In [42]: print(l)
['s', 'u', 'd', 'a', 'r', 's', 'h', 'a', 'n', '#', '*', '&', 'I', 'n', 'd', 'i', 'a', 'B', 'h', 'a', 'r', 'a',
 't', 'h', '(', ')', 'K', 'a', 'r', 'n', 'a', 't', 'a', 'k', 'a', '$', 'P', 'u', 'n', 'j', 'a', 'b', '#', 'T', 'a', 'm',
 'i', 'l', ' ', 'N', 'a', 'd', 'u']

In [43]: r=[]

In [44]: for p in l:
...:     if p is in alpha:
...:         r.append(p)
...:
File ~\ipython-input-44-2ce62dd5ea6*, line 2
if p is in alpha:
    ^
SyntaxError: invalid syntax

In [45]: for p in l:
...:     if p in alpha:
...:         r.append(p)
```

```
In [46]: print(r)
['s', 'u', 'd', 'a', 'r', 's', 'h', 'a', 'n', 'i', 'n', 'd', 'i', 'a', 'B', 'h', 'a', 'r', 'a', 't', 'h', 'K', 'a', 'r', 'n', 'a', 't', 'a', 'k', 'a', 'p', 'u', 'n', 'j', 'a', 'b', 'T', 'a', 'm', 'i', 'l', 'M', 'a', 'd', 'u']

In [47]: print(alpha)
('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z')

In [48]: l=list(range(10))

In [49]: t=tuple(l)
Traceback (most recent call last):

  File "<ipython-input-49-a2b90ef83307>", line 1, in <module>
    t=list(tuple(l))
TypeError: 'int' object is not iterable

In [50]: t=tuple(range(10))

In [51]: print(l)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [52]: print(t)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

In [53]: l.__sizeof__()
Out[53]: 120

In [54]: |

In [40]: l=list(range(10))

In [49]: t=tuple(l)
Traceback (most recent call last):

  File "<ipython-input-49-a2b90ef83307>", line 1, in <module>
    t=list(tuple(l))
TypeError: 'int' object is not iterable

In [50]: t=tuple(range(10))

In [51]: print(l)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [52]: print(t)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

In [53]: l.__sizeof__()
Out[53]: 120

In [54]: t.__sizeof__()
Out[54]: 104

In [55]: print("When we are sure of the list that we are handling and we are also sure that it never changes, then use a tuple")
When we are sure of the list that we are handling and we are also sure that it never changes, then use a tuple

In [56]: lists, dictionaries, set, tuple |
```

Let us say, what is set of s, you see it takes individually and converts it like this. So, let us say s equals set of s, which means your s is no more a string like this, but it becomes a set and that again gets assigned to s and the string s is lost. As you can see, if you print s, you will have, you see it is not even ordered, a, b, c, d in some order it is. But I want this as a tuple. So, alpha equals tuple of s, so print alpha, so it has all the characters.

Maybe I will do one thing. I do not like this in a random ordering. I want the right ordering. It is a very easy thing to do. What I will do is alpha equals, what is that s is equal to, what did I say, string ASCII letters, I am done. And from this, I will say alpha is equal to tuple of list of s. List of s makes it, what is list of s, let me show you. It makes it like this, print list of s is like this.

I tend to repeat the commands just so that I reiterate and it is easy on your minds. I can of course go fast. But it does not make sense given that I should match your thinking and your absorption rate. That is why I tend to be repetitive. Some people who know stuff already or good at Python might find it boring as always, I am sorry. I should look at the average of the intended audience. So far, so good. So, I create this.

Now, you know what is an alpha, print alpha. You will get the tuple a, b, c, d, e, f, g, h. Now, this is fixed. You cannot change this. Can you? You cannot change this. Now, I am going to use this for a small application. And what is that? That is, I will create a new string, namely x is equal to, Sudarshan hash percentage ampersand so on and so forth and then I will say India and then I will say Bharath and then I will put brackets and then I will say Karnataka and then Punjab, so on and so forth, Tamil Nadu, I will even include a space here, Tamil Nadu and so on.

So, now, what I want to do is, maybe I will change this to a list, so that it is easy in our minds. I now want to remove anything that is not here. I want to remove all the unwanted symbols and retain only the valid alphabets. So, how do you do that? It is going to be a very simple, straightforward code. Let me write that here itself. Bear with me, I am not going to open a file and then type it. You are now familiar with terminal. So, very straightforward terminal is like a calculator. In calculator, you do not store anything.

You go ahead and then simply type it. So, that is what I am going to do. I am going to simply go ahead and then type my code. What do I do for, let us say, p in, I will not say x in, because x is used already, for p in l for every single letter you have, I will first initialize an array r. What I will do is for p in l, so Python is as simple as English as you know. I am iterating through every single entry of l here. What I will do is I will pick only the valid letters and put them into l. If p is in alpha, then I will append that to r.

Let us spend a moment watching this. I mean, my intention is also to sort of get you familiarized with programming, programmatic thinking, thinking programmatically. For p in l, p is just a variable. It iterates through the entire list l which is basically this. Initially p becomes s and then p becomes u and then p becomes d and then p becomes a and so forth, so on and so forth. And every time you check if p is in alpha. Is in is a valid way in which you, in Python you say if p is an alpha, true it gets inside. If it is not in alpha, it does not get inside. I am sure you are familiar with if now. You are also now familiar with is in.

So, this should be, I am sorry, I made a mistake. There is no is in there, there is only in there. Never mind, mistakes do happen. You all make mistakes so do I. Let me console myself by saying that. Yes, there is no error here. Let me now say print r and you get all the letters Sudarshan, India, Bharath and then what did I, and hedge, did I type hedge somewhere, India, Bharat and Karnataka and so on, everything is here. And even the special symbols including the space is removed, because space was not part of this alpha. The question is you used alpha as a lookup to remove unwanted letter.

Now, alpha is fixed, you see. This particular alpha is fixed. You will never change this, will you? So, this is alphabets, all alphabets. You do not want to change this. You do not want to change alpha. It is fixed. In such instances when you do not want to change something, when you fix it, you use tuples. Now, you can ask me, why not use lists, because accidentally you might remove something from the list. Is that all a why you would use tuples. There is one hidden agenda, why you use tuples. And I will handle it for you right now just observe.

Let us say l equals, so we have this l here, let me do something else, a equals list of, let me use l itself is in the mind, list of range of 10, t equals list of tuple of 10 and you see, what went wrong there, tuple of range of 10, my bad, tuple of range of 10, you have your list, you have your t. What was the question? The question was, why do you even care for a tuple except that it cannot be changed. We can as well use a list. There is a hidden benefit of this. And that is, you see, l underscore, underscore, remember this, size of l is 120. There is a beautiful function called size of. It will tell you how much memory size l is using.

What is your guess if I said size of t? Boom, that is 104. Do you see the point here? Do you see the point we discussed with a cupboard with the wheels and not with the wheels? Obviously, I would say the cupboard without the wheels should cost me less. Here is a facility where you do not want to remove it. This is stagnant. This is constant. You will not move it. Here you may have to add, you may have to remove, so your computer Python uses your computer to store it in a slightly different way so that it is flexible.

By with flexibility comes price probably that is why it says 120, it says 104. In fact, it is out of the scope of our discussion to actually get into the details of how exactly it is stored. But you can look up online with a little smart as you will actually understand what is happening here. So good, so far, so good.

You now understood why list, why dictionary, why set, why tuple. Whenever you are very sure, let me just print that we imprint that on your minds, when we are sure of the list that we are handling and we are also sure that it never changes then use a tuple. So, take this, put this in your mind, remember it and please use tuples whenever you have, imagine you have a very, very big set of data and the data remains the same, you will never change it. So, you can at least reduce the memory that you use.

So, word of caution, do not bring your head much about the lesson tuples and even dictionaries and sets and what about not. I mean, need based usage. Whenever you feel like using something, use it. If something is not working because it is the huge data that you are crunching, if it is huge data that you are crunching, do not worry, it will take some time for you to reach there. But when you reach there, you would have automatically matured to what should be used where. You can always look up on the Internet what is the best way to get something done.

So, as of now just for the completeness sake, we have taught you the lists, the dictionaries, set and a tuple. Remember what is what that should do for now. Good.

