

The screenshot shows a video player interface for 'GO CLASSES'. The main content area displays a slide with the title 'Using Lists' in large white font. Below the title is a bulleted list of points:

- We need an easy way to hold individual data items without needing to make lots of variables
  - Making `num1`, `num2`, ..., `num99`, `num100` is time-consuming and impractical
- Instead, we can use a *list* to hold our data
  - A list is a **data structure**: something that holds multiple pieces of data in one structure

At the bottom of the slide, there is a red footer bar with the 'www.goclasses.in' logo and URL.

The screenshot shows a video player interface for 'GO CLASSES'. The main content area displays a slide with the title 'List Syntax' in large blue font. Below the title, a text block states: 'We use **square brackets** to set up a list in Python.' Below this text, two examples of Python code are shown:

```
a = [ ] # empty list
b = [ "uno", "dos", "tres" ] # list with 3 strings
```

At the bottom of the slide, there is a red footer bar with the URL 'https://www.cs.cmu.edu/~15110-f20/slides/week5-1-lists.pdf' and a video control bar showing the current time as 02:28 and the total duration as 1:22:10.

The screenshot shows a video player interface for 'GO CLASSES'. In the top right corner, there is a video feed of a male host wearing glasses and a black shirt, with a microphone in front of him. The word 'Host' is visible below the video frame. The main content area displays a slide with text and code snippets. The text on the slide includes:

- Equivalent to arrays in other languages.
- Lists are mutable, i.e., a list can be changed without having to create a new list object

Below the text is a code block:

```
>>> L = [25,1,2,15]
>>> L
[25, 1, 2, 15]
```

At the bottom of the slide, there is a URL: [https://www.purdue.edu/hla/sites/varalalab/wp-content/uploads/sites/20/2018/03/Lecture\\_9.pdf](https://www.purdue.edu/hla/sites/varalalab/wp-content/uploads/sites/20/2018/03/Lecture_9.pdf)

The screenshot shows a video player interface for 'GO CLASSES'. In the top right corner, there is a video feed of a male host wearing glasses and a black shirt, with a microphone in front of him. The word 'Host' is visible below the video frame. The main content area displays a slide with a section title and a list:

## Numbering in Lists

- Lists don't start counting from 1
  - They start counting from 0!
- Lists with n elements are numbered from 0 to n-1
  - The list below has 5 elements, and is numbered from 0 to 4

Below the list is a diagram showing a horizontal row of five empty boxes, each labeled with a number from 0 to 4 above it:

0	1	2	3	4

At the bottom of the slide, there is a logo for 'www.goclasses.in'.

The screenshot shows a video player interface. At the top left is the 'GO CLASSES' logo. On the right is a video frame showing a man with glasses and a microphone, identified as the 'Host'. The video title 'List Syntax' is centered above the video frame. Below the video frame is a control bar with icons for volume, pause, and other media controls. The video duration is listed as 1:20:40.

## List Syntax

- Use `[ ]` to assign initial values (*initialization*)  
`myList = [1, 3, 5]  
words = ["Hello", "to", "you"]`
- And to refer to individual elements of a list  
`>>> print(words[0])  
Hello  
>>> myList[0] = 2`

The screenshot shows a video player interface. At the top left is the 'GO CLASSES' logo. On the right is a video frame showing the same man from the previous screen. The video content discusses lists as sequence objects. Below the video frame is a control bar with icons for volume, pause, and other media controls. The video duration is listed as 1:17:45.

- List is a more general sequence object that allows the individual items to be of different types.

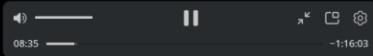
```
c = [ 1, "woof", 4.5 ] # lists can have mixed types
```

[https://www.purdue.edu/hla/sites/varalalab/wp-content/uploads/sites/20/2018/03/Lecture\\_9.pdf](https://www.purdue.edu/hla/sites/varalalab/wp-content/uploads/sites/20/2018/03/Lecture_9.pdf)

**GO CLASSES**



In [10]: mylist = ["hello", "this", "is", "python", "class"]  
mylist  
Out[10]: ['hello', 'this', 'is', 'python', 'class']  
  
In [11]: mylist[2]  
Out[11]: 'is'  
  
In [14]: mylist[3]  
Out[14]: 'python'  
  
In [15]: mylist[3][0]  
Out[15]: 'p'



**GO CLASSES**



In [18]: listb = ["hello", 3, 4.5, [1,2,3]]  
listb  
Out[18]: ['hello', 3, 4.5, [1, 2, 3]]

# Properties of a List

- Heterogeneous (multiple data types!)
- Contiguous (all together in memory)
- Ordered (numbered from 0 to n-1)

Host

www.goclasses.in

The diagram illustrates list indexing with two rows of boxes representing elements.

**Forward Indexing:** The top row shows a sequence of boxes indexed from 0 to 4 in yellow, followed by a red box at index 5, and another red box at index 6. A red arrow points from the text "index out of range" to the red box at index 7. A blue box labeled "List" contains the sequence 1, 2, 3, 4, 5.

**Reverse Indexing:** The bottom row shows a sequence of boxes indexed from -8 to -1 in red, followed by a yellow box at index -5. A red arrow points from the text "index out of range" to the yellow box at index -5. A blue box labeled "List" contains the sequence 1, 2, 3, 4, 5.

www.goclasses.in

13:15 -1:11:23

The screenshot shows a video player interface for 'GO CLASSES'. The video content is a lesson titled 'Python Lists'. A male host is visible in the top right corner. The main area displays a bulleted list of points about negative indexing:

- Can use negative index to work back from end of list
  - What?!
- Bring me the strangeness!
  - letters = ['a', 'b', 'c', 'd', 'e']
  - letters[-1] is 'e'
  - letters[-2] is 'd'
  - letters[-5] is 'a'
  - For indexes, think of `-x` as same as `len(list)-x`
  - letters[-1] is same as letters[len(letters)-1]

Below the video player, there is a browser-like control bar with a progress bar, volume controls, and other media controls.

The screenshot shows a video player interface for 'GO CLASSES' with a title bar reading 'Lists vs. Arrays'. A male host is visible in the top right corner. The main content area contains a comparison between arrays and Python lists:

Many programming languages have an **array** type. Python doesn't have native arrays (though some Python libraries add arrays).

A diagram illustrates the difference in indexing between arrays and lists. It shows a horizontal array of 10 slots, indexed from 0 to 9. An arrow points to the slot at index 8, labeled 'Element (at index 8)'. Another arrow points to the first slot, labeled 'First index'. A double-headed arrow below the array is labeled 'Array length is 10'.

Arrays are:

- homogeneous (all elements are of the same type)
- fixed size
- permit very fast access time

Python lists are:

- heterogeneous (can contain elements of different types)
- variable size
- permit fast access time

**Slicing:**

**Slicing syntax:** list [ *start*:*end* ]

index of first character      1 + index of last character

www.goclasses.in

20:33 - 1:04:05

a = [10, 20, 30, 40, 50]

a[1:3]  $\Rightarrow$  20, 30

a[-4:-1]  $\Rightarrow$  20, 30, 40

a[1:]  $\Rightarrow$  20, 30, 40, 50

www.goclasses.in

21:36 - 1:03:02

The screenshot shows a video player interface for 'GO CLASSES'. The title 'Converting Strings Into Lists' is displayed at the top. A small video window in the top right corner shows a male host wearing glasses and a black shirt, with his right hand raised. Below the title, there is a code snippet and a bulleted list:

```
>>> a = "hello"
>>> list(a)
['h', 'e', 'l', 'l', 'o']
```

- The **list** function can convert a string to a list.
- Always works.
- **Very handy** if we want to manipulate a string's contents and create new strings based on them.

At the bottom of the screen, a URL is visible: [https://crystal.uta.edu/~athitsos/courses/cse1310\\_summer2013/lectures/7a\\_strings.pdf](https://crystal.uta.edu/~athitsos/courses/cse1310_summer2013/lectures/7a_strings.pdf).

The screenshot shows a video player interface for 'GO CLASSES'. The title 'STRINGS to LISTS' is displayed at the top. A small video window in the top right corner shows the same male host from the previous video. Below the title, there is a bulleted list:

- Convert **string to list** with **list(s)**
  - Every character from s is an element in a list

Below the list, a code snippet is shown in a box:

```
In [1]: word = "GATE DA"
list(word)
Out[1]: ['G', 'A', 'T', 'E', ' ', 'D', 'A']
```

At the bottom of the screen, a URL is visible: [https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6\\_100l\\_f22\\_lec10.pdf](https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6_100l_f22_lec10.pdf). A video control bar is visible at the bottom, showing a play button, volume controls, and a progress bar indicating the video is at 24:05 of 1:00:33.

A screenshot of a Jupyter Notebook interface within a Chrome browser window. The notebook shows the following Python code:

```
In [20]: s = "gate da"
s
Out[20]: 'gate da'

In [23]: li = list(s)
In [24]: li
Out[24]: ['g', 'a', 't', 'e', ' ', 'd', 'a']

In [ ]: str(li)
```

The video player at the bottom indicates the video is at 25:16 and ends at 59:22.

A screenshot of a Jupyter Notebook interface within a browser window. The notebook shows the same Python code as the first screenshot, but with handwritten annotations:

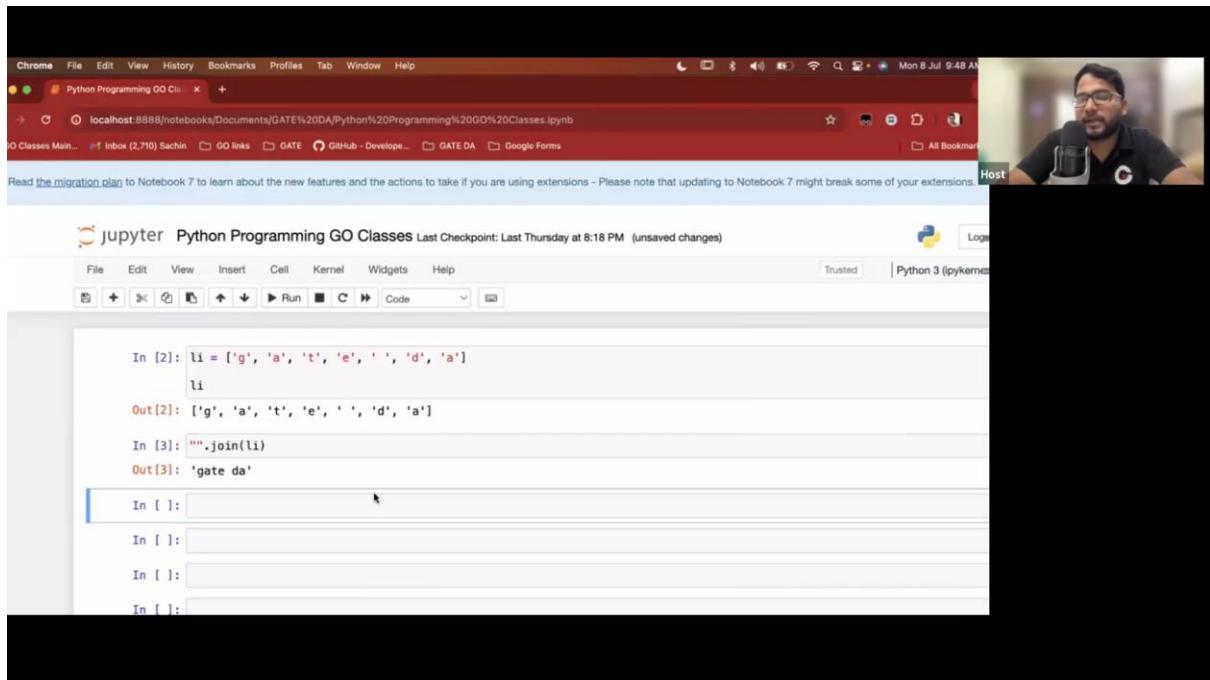
list to string

```
In [20]: s = "gate da"
s
Out[20]: 'gate da'

In [23]: li = list(s)
In [24]: li
Out[24]: ['g', 'a', 't', 'e', ' ', 'd', 'a']

In [25]: str(li) ← it doesn't work
Out[25]: "[‘g’, ‘a’, ‘t’, ‘e’, ‘ ’, ‘d’, ‘a’]"
```

Handwritten notes: "list to string" is written above the code. An arrow points from the handwritten note "it doesn't work" to the output cell of the str(li) command. The video player at the bottom indicates the video is at 26:57 and ends at 57:41.

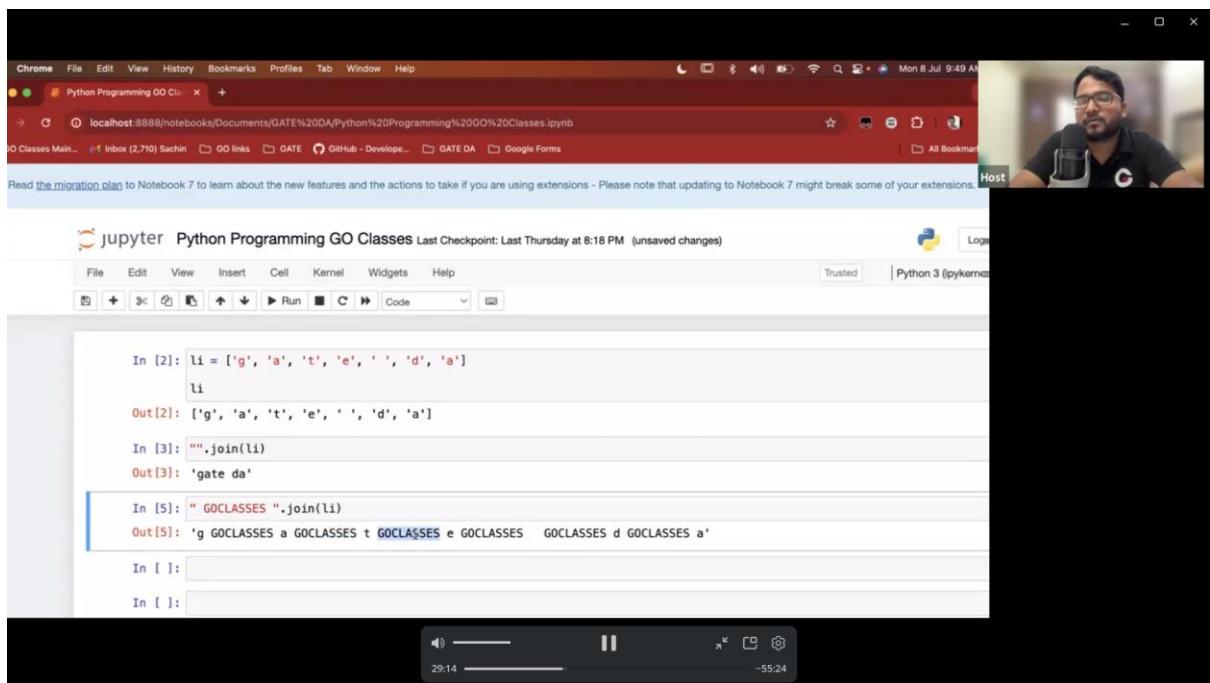


A screenshot of a video conference call. On the right, a man with glasses and a black polo shirt is visible. He is speaking into a microphone. The background is slightly blurred. On the left, a computer screen displays a Jupyter Notebook interface. The title bar says "jupyter Python Programming GO Classes Last Checkpoint: Last Thursday at 8:18 PM (unsaved changes)". The notebook contains the following code:

```
In [2]: li = ['g', 'a', 't', 'e', ' ', 'd', 'a']
li
Out[2]: ['g', 'a', 't', 'e', ' ', 'd', 'a']

In [3]: ''.join(li)
Out[3]: 'gate da'

In [ ]:
```



A screenshot of a video conference call, continuing from the previous one. The man on the right is still speaking. The Jupyter Notebook interface on the left now shows this additional command:

```
In [5]: " GOCLASSES ".join(li)
Out[5]: 'g GOCLASSES a GOCLASSES t GOCLASSES e GOCLASSES GOCLASSES d GOCLASSES a'
```

A screenshot of a Jupyter Notebook running in a Chrome browser. The notebook shows the following code execution:

```
In [2]: li = ['g', 'a', 't', 'e', ' ', 'd', 'a']
        li
Out[2]: ['g', 'a', 't', 'e', ' ', 'd', 'a']

In [3]: ''.join(li)
Out[3]: 'gate da'

In [6]: "_".join(li)
Out[6]: 'g_a_t_e_ _d_a'
```

The video player interface at the bottom indicates the video is at 29:24 of a 55:14 minute duration.

LISTS to STRINGS

- Convert a **list of strings back to string**
- Use `' .join(L)` to turn a **list of strings into a bigger string**
- Can give a character in quotes to add char between every element

```
L = ['a', 'b', 'c']           → L is a list
A = ' '.join(L)              → A is "abc"
B = '_'.join(L)              → B is "a_b_c"
```

---

www.goclasses.in  
30:05 - 54:33

Chrome File Edit View History Bookmarks Profiles Tab Window Help

localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb

Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions.

jupyter Python Programming GO Classes Last Checkpoint: Last Thursday at 8:18 PM (unsaved changes)

In [7]: li = ["Everybody", "is", "looking", "forward", "to", "the", "weekend"]  
li  
Out[7]: ['Everybody', 'is', 'looking', 'forward', 'to', 'the', 'weekend']

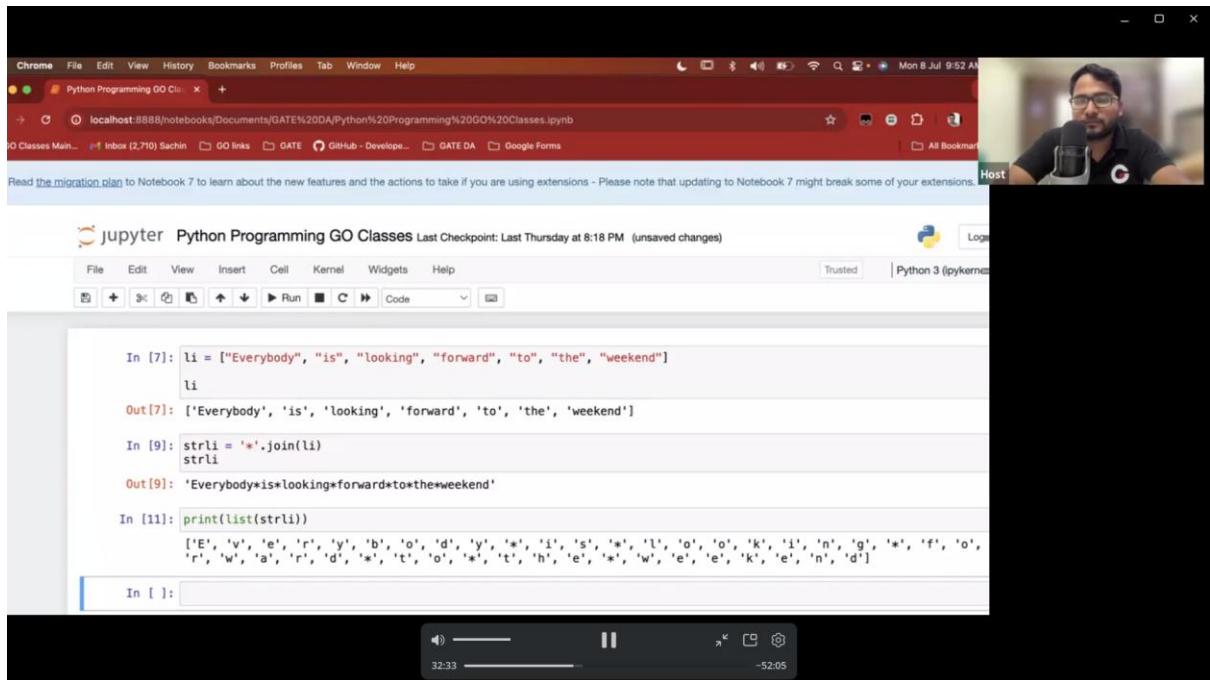
In [9]: strli = '\*' .join(li)  
strli  
Out[9]: 'Everybody\*is\*looking\*forward\*to\*the\*weekend'

In [11]: print(list(strli))  
['E', 'v', 'e', 'r', 'y', ' ', 'i', 's', ' ', 'l', 'o', 'o', 'k', 'i', 'n', 'g', ' ', 'f', 'o', 'r', 'w', 'a', 'r', 'd', ' ', '\*', 't', 'o', ' ', 't', 'h', 'e', ' ', 'w', 'e', 'e', 'k', 'e', 'n', 'd']

In [ ]:

Host

32:33 -52:05

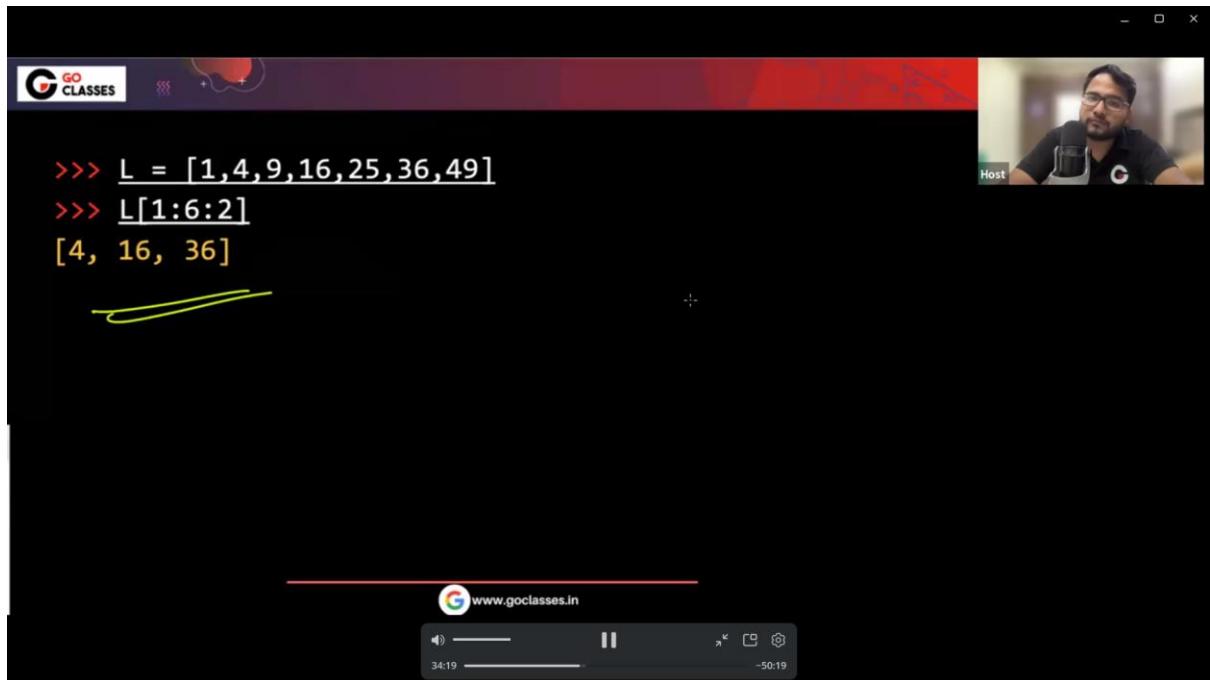


GO CLASSES

>>> L = [1,4,9,16,25,36,49]  
>>> L[1:6:2]  
[4, 16, 36]

www.goclasses.in

34:19 -50:19



The screenshot shows a video player interface for 'GO CLASSES'. On the left, a terminal window displays Python code being run, showing list slicing operations on a list 'L' containing integers from 1 to 49. On the right, a video feed of a male host wearing glasses and a dark shirt is visible, sitting in front of a microphone. The video player has a progress bar at the bottom indicating the video is at 34:55 of 49:43.

```
>>> L = [1,4,9,16,25,36,49]
>>> L[2:4]
[9, 16]
>>> L[1:]
[4, 9, 16, 25, 36, 49]
>>> L[:5]
[1, 4, 9, 16, 25]
>>> L[1:6:2]
[4, 16, 36]
>>> L[::-1]
[49, 36, 25, 16, 9, 4, 1]
>>> L[:]
[1, 4, 9, 16, 25, 36, 49]
```

Similarly, `step` may be a negative number:



```
a = [1,2,3,4,5,6,7,8,9,10]

a[::-1]      # all items in the array, reversed
a[1::-1]    # the first two items, reversed
a[:-3:-1]   # the last two items, reversed
a[-3::-1]   # everything except the last two items, reversed
```

---



`append()` method

- Can add elements to end of list with `.append`

```
alist = [10, 20, 30]
alist.append(40)
```

alist → 

10	20	30	40
[10, 20, 30, 40]			



The screenshot shows a video player interface for 'GO CLASSES'. The title 'append() method' is displayed at the top. A video feed of a host wearing glasses and a black shirt is visible on the right. The main content area contains a bulleted list and some code examples.

- Can add elements to end of list with `.append`

```
alist = [10, 20, 30]
alist.append(40)
alist.append(50)
new_list = []
new_list.append('a')
new_list.append(4.3)
```

new\_list → 

'a'	4.3
['a', 4.3]	

  
alist → 

10	20	30	40	50
[10, 20, 30, 40, 50]				

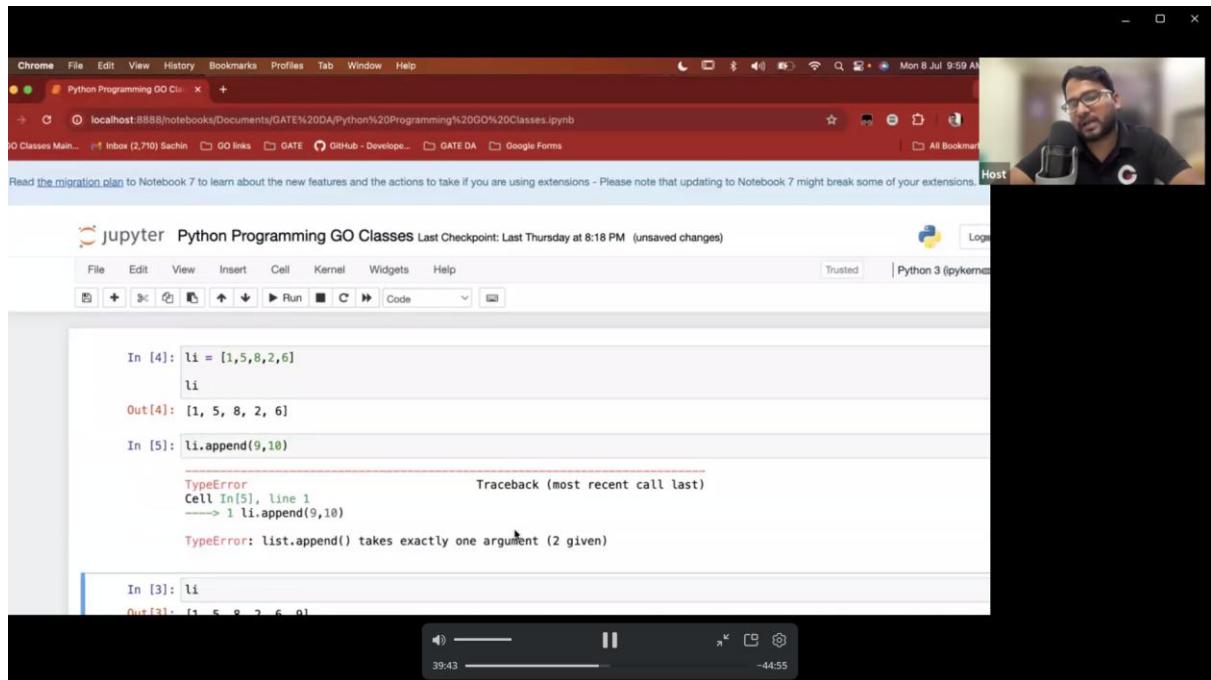
Below the video player are standard media controls: back, forward, volume, and a progress bar indicating the video is at 37:16 of a total duration of 47:22.

The screenshot shows a video player interface for 'GO CLASSES'. The title 'extend() method' is displayed at the top. A video feed of a host wearing glasses and a black shirt is visible on the right. The main content area contains a bulleted list and some code examples.

- Function: `list.extend(other_list)`
  - Adds all element from other list to list that function is called on

```
>>> list1 = [1, 2, 3]
>>> list2 = [4, 5]
>>> list1.extend(list2)
>>> list1
[1, 2, 3, 4, 5]
```

Below the video player are standard media controls: back, forward, volume, and a progress bar indicating the video is at 38:13 of a total duration of 46:25. A watermark for 'www.goclasses.in' is visible at the bottom of the screen.



## Summary of the difference:

Code	Action	Result
<code>my_list.extend([4])</code>	Adds each <i>element</i> of the provided list <code>[4]</code> to <code>my_list</code> .	If <code>my_list</code> was <code>[1, 2, 3]</code> , it becomes <code>[1, 2, 3, 4]</code> .
<code>my_list.extend(4)</code>	Attempts to iterate over the integer <code>4</code> .	Raises a <code>TypeError</code> because an integer is not iterable.
<code>my_list.append(4)</code>	Adds the entire integer <code>4</code> as a <i>single element</i> to <code>my_list</code> .	If <code>my_list</code> was <code>[1, 2, 3]</code> , it becomes <code>[1, 2, 3, 4]</code> .

**GO CLASSES**



In [10]: li = [1,5,8,2,6]  
li + [9,10]  
print(li)  
[1, 5, 8, 2, 6]

In [11]: a = 5  
a+3  
print(a)  
5

Host

44:13 - 40:25

**GO CLASSES**



In [12]: li = [1,5,8,2,6]  
alist = li + [9,10]  
print(li) #first list  
print(alist) #new list created  
[1, 5, 8, 2, 6]  
[1, 5, 8, 2, 6, 9, 10]

Host

46:18 - 38:20

A screenshot of a Jupyter Notebook interface running in a Chrome browser window. The title bar shows "Python Programming GO Classes". The notebook content area displays two code cells:

```
In [15]: li = [1,5,8,2,6]
li.append([9,10])
li
Out[15]: [1, 5, 8, 2, 6, [9, 10]]
```

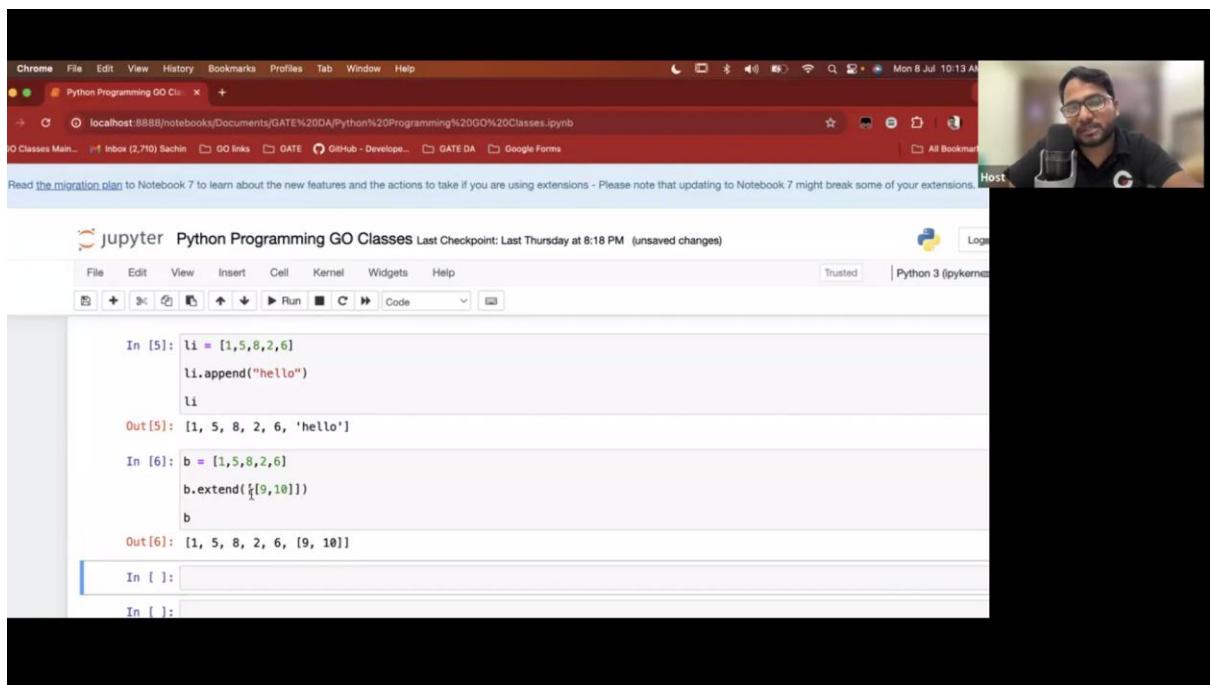
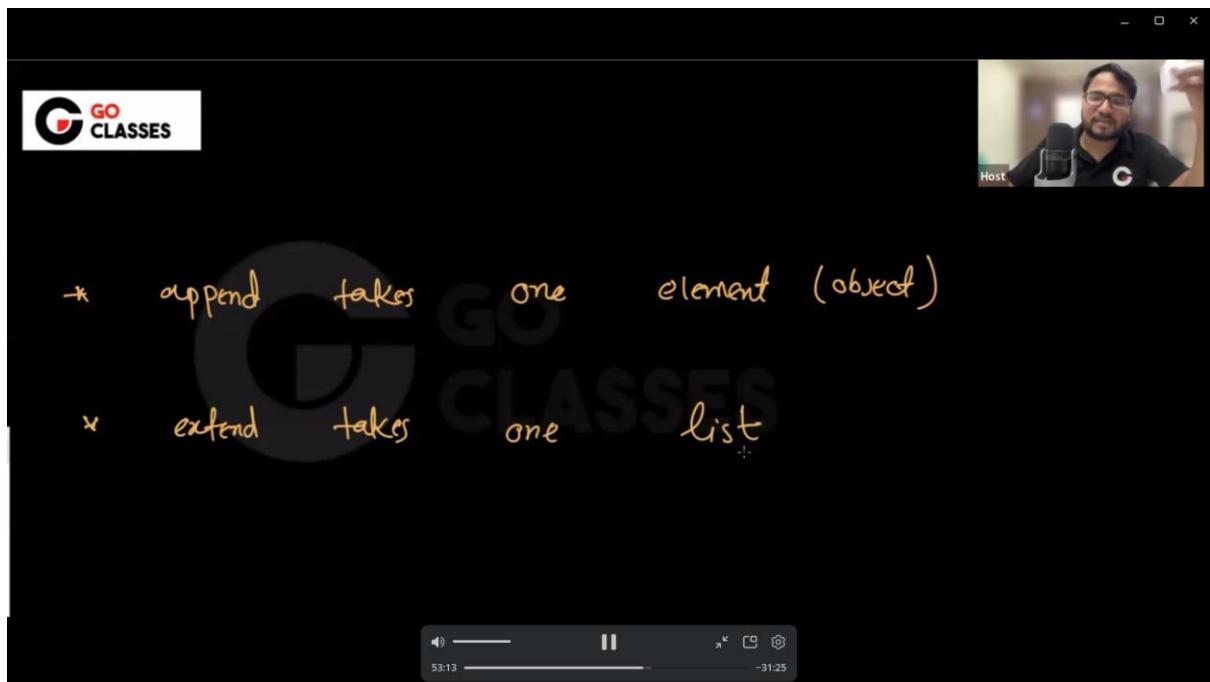
```
In [14]: b = [1,5,8,2,6]
b.extend([9,10])
b
Out[14]: [1, 5, 8, 2, 6, 9, 10]
```

The "Host" video feed is visible in the top right corner.

A screenshot of a Jupyter Notebook interface running in a browser window. The title bar shows "GO CLASSES". The notebook content area displays one code cell:

```
In [5]: li = [1,5,8,2,6]
li.append("hello")
li
Out[5]: [1, 5, 8, 2, 6, 'hello']
```

A yellow arrow points from the word "hello" in the code to the word "object" in the output, with the text "one element object" written next to it. The "Host" video feed is visible in the top right corner. A video player control bar is at the bottom.



The screenshot shows a video player interface for a Go Classes lesson. The top bar has the 'GO CLASSES' logo and a search bar. On the right, there's a video feed of a host wearing glasses and a black shirt, with the word 'Host' below it. The main content area has a light gray background. On the left, there are navigation icons: a back arrow, a forward arrow, a refresh circle, and a checkmark. A large number '5878' is displayed prominently. Below it, two code snippets are shown in a monospaced font:

```
>>> x = [1, 2, 3]
>>> x.append([4, 5])
>>> print(x)
[1, 2, 3, [4, 5]]
```

**5878**

`.append()` appends a **single object** at the end of the list:

```
>>> x = [1, 2, 3]
>>> x.extend([4, 5])
>>> print(x)
[1, 2, 3, 4, 5]
```

`.extend()` appends **multiple objects** that are taken from inside the specified iterable:

At the bottom, there's a red horizontal bar with the Go Classes logo and the URL 'www.goclasses.in'. Below the video player, there are standard media controls: play/pause, volume, and a progress bar showing '54:53' and '29:46'.

```
li=[1,2] li.extend([3,4],[5,6]) print(li)
```

### AI Overview

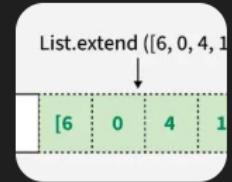


En



Listen

The code will raise a `TypeError` because the `extend()` method can only take a single iterable as an argument, not multiple separate iterables like `[3,4]` and `[5,6]`. To achieve the desired output of `[1, 2, 3, 4, 5, 6]`, you should use two separate `extend()` calls or pass a single list containing both sub-lists, like `li.extend([3, 4, 5, 6])`.



### Corrected code

python

```
li = [1, 2]
li.extend([3, 4])
li.extend([5, 6])
print(li)
```

### Output

```
[1, 2, 3, 4, 5, 6]
```

**GO CLASSES**

## Adding elements to a List

Host

- Here are a few examples that show how to use the list `.append()` method vs `+` operator to add items to the end of an existing list

```
In [8]: numList = [1, 2, 3, 4, 5]
In [9]: numList + [6]           list concatenation
Out[9]: [1, 2, 3, 4, 5, 6]    this is a new list!
In [10]: numList # numList has not changed
Out[10]: [1, 2, 3, 4, 5]
In [12]: numList.append(6)     list append, notice dot notation
In [14]: numList # numList has been updated to include 6
Out[14]: [1, 2, 3, 4, 5, 6]
```

**GO CLASSES**

## Adding elements to a List

Host

- Here are a few examples that show how to use the list `.append()` method vs `+` operator to add items to the end of an existing list

```
In [8]: numList = [1, 2, 3, 4, 5]
In [9]: numList + [6]           list concatenation
Out[9]: [1, 2, 3, 4, 5, 6]    this is a new list!
In [10]: numList # numList has not changed
Out[10]: [1, 2, 3, 4, 5]
In [12]: numList.append(6)     list append, notice dot notation
In [14]: numList # numList has been updated to include 6
Out[14]: [1, 2, 3, 4, 5, 6]
```

s = "hello"

s = uppercase

print(s)

hell0

The screenshot shows a video player interface for 'GO CLASSES'. The title 'Basic List Operations' is displayed at the top. A small video thumbnail of a host wearing glasses and a black shirt is visible in the top right corner. Below the title, a snippet of Python code is shown:

```
a = [ 1, 2 ] + [ 3, 4 ] # concatenation - [ 1, 2, 3, 4]
b = [ "a", "b" ] * 3 # repetition - [ "a", "b", "a", "b", "a", "b" ]
c = [ 1, 3 ] < [ 2, 4 ] # comparison/equality - True
d = 4 in [ "a", "b", 1, 2 ] # membership - False
```

At the bottom of the screen, there is a watermark for 'www.goclasses.in'.

The screenshot shows a Jupyter Notebook cell with the following code and output:

```
Out[7]: [1, 5, 8, 2, 6, 7, [9, 10]]
In [8]: [1,3] < [2,4]
Out[8]: True
In [9]: [1,3] < [2,2]
Out[9]: True
In [10]: [1,2,3,4] < [1,2,5,0]
Out[10]: True
```

A video thumbnail of the host is visible in the top right corner. At the bottom of the screen, there is a media control bar with volume, play/pause, and other controls, showing a timestamp of 59:52 and a duration of -24:46.

The screenshot shows a video player interface for 'GO CLASSES'. In the top right corner, there is a video thumbnail of a man with glasses and a black shirt, labeled 'Host'. The main area displays a Python code editor with the following content:

```
starks = ["Ned", "Arya", "Bran", "Sansa"]

✓ "Ned" in starks
✗ "Sansa" in starks[1:3]
✓ len(starks[1:4]) == 3
✗ "Arya" in (starks + ["Jon"])[2:]
✗ len(starks[1:2] * 4) == 8
```

Below the code editor, a URL is visible: <https://facultyweb.cs.wwu.edu/~wehrwes/courses/csci141/fall2020/L20.pdf>. At the bottom of the video player, there is a control bar with a play button, volume controls, and a progress bar indicating the video is at 1:02:08 of a total duration of 22:30.

Yes, `len(st[1:4]) == 3` evaluates to `True` [1]. ⓘ

Here is a breakdown of why:

1. `st` is the list `['ab', 'bcc', 'ddd', 'ghj']`.
2. The slice `st[1:4]` takes elements starting from index 1 (inclusive) up to, but not including, index 4.
3. The elements selected are:
  1. Index 1: `'bcc'`
  2. Index 2: `'ddd'`
  3. Index 3: `'ghj'`
4. The resulting sublist is `['bcc', 'ddd', 'ghj']`.
5. The length of this sublist is 3. ⓘ

Therefore, `len(st[1:4])` is 3, and the comparison `3 == 3` is `True` [1].

No, `len(st[1:2]*4)` does not equal 10. The result is 4.

Here is the breakdown:

1. `st[1:2]` : This is a list slice that extracts the element at index 1 (which is `'bcc'`) into a new list. The result is `['bcc']`. Note that slicing a list, even for a single element, returns a list, not the element itself.
2. `st[1:2] * 4` : This repeats the list `['bcc']` four times through concatenation. The result is `['bcc', 'bcc', 'bcc', 'bcc']`.
3. `len(['bcc', 'bcc', 'bcc', 'bcc'])` : The `len()` function returns the number of items in the list, which is 4.

Therefore, `len(st[1:2]*4)` is 4, which is not equal to 10.

Which of the following is true?

```
starks = ["Ned", "Arya", "Bran", "Sansa"]
```

a) `"Ned" in starks` ✓  
b) `"Sansa" in starks[1:3]` ✗  
c) `len(starks[1:4]) == 3` ✓  
d) `"Arya" in (starks + ["Jon"])[2:]` ✗  
e) `len(starks[1:2] * 4) == 8`

Priya → [ Arya, Arya, Arya, Arya ]

<https://facultyweb.cs.wvu.edu/~wehrwes/courses/cs3519/lectures/L20/L20.pdf>

1:02:57 -21:41

The screenshot shows a video player interface. At the top, there's a header bar with the 'GO CLASSES' logo and some decorative icons. On the right side of the header, there's a small video thumbnail of a man wearing glasses and a black shirt, with the word 'Host' below it. The main content area has a dark background with a large, semi-transparent watermark-like text 'ES' in the center. Below this, the title 'Two-Dimensional Lists' is displayed in a large, white, sans-serif font. At the bottom of the slide, there's a navigation bar with a 'www.goclasses.in' logo, volume controls, a play/pause button, and a progress bar indicating the video is at 1:05:45 of a total duration of 18:53.

This screenshot is similar to the one above, showing a video player interface. The header bar includes the 'GO CLASSES' logo and a 'Host' thumbnail. The main content area features the title 'Two-Dimensional Lists: A Grid' in a large, white, sans-serif font. Below the title, there's a bulleted list: '• May help to think of 2D lists as a grid'. Underneath the list, there's a line of code: 'twoD = [ [1,2,3], [4,5,6], [7,8,9] ]'. To the right of the code, there's a 3x3 grid table with the following data:

1	2	3
4	5	6
7	8	9

At the bottom of the slide, there's a navigation bar with a 'www.goclasses.in' logo, volume controls, a play/pause button, and a progress bar indicating the video is at 1:05:45 of a total duration of 18:53.

**GO CLASSES**

```
In [11]: li = [[1,2,3], [4,5,6], [7,8,9]] #list of lists  
li  
Out[11]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
  
In [12]: li[-1][1:]  
Out[12]: [8, 9]
```

## Two-Dimensional Lists: A Grid

- You access an element by the index of its row, then the column
  - Remember – indexing starts at 0!

www.goclasses.in

**GO CLASSES**

## MultiDimensional Lists

```
my2DList = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]]
```

Host

www.goclasses.in

1:07:40 - 16:58

**GO CLASSES**

```
In [15]: li = [[1,2,3], [4,5,6], [7,8,9]] #list of lists  
        bi = li  
        bi[0][0] = 900  
  
        li
```

Out[15]: [[900, 2, 3], [4, 5, 6], [7, 8, 9]]

Host

www.goclasses.in

1:11:02 - 13:36

Shallow copy

The screenshot shows a video player interface. At the top left is the 'GO CLASSES' logo. On the right is a video frame showing a man with glasses and a beard, identified as the 'Host'. Below the video frame is a small 'Host' label. The main content area has a dark background with white text. The title 'MUTABILITY' is at the top. Below it is a bulleted list:

- Lists are **mutable**!
- Assigning to an element at an index **changes** the value

Following the list is some Python code:

```
L = [2, 4, 3]
L[1] = 5
```

A callout diagram illustrates this: a variable 'L' points to a cloud-like box containing '[2, 5, 3]'. A yellow arrow points from the box to the text 'different from strings and tuples!'. Below the video player is a URL: [https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6\\_100l\\_f22\\_lec10.pdf](https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6_100l_f22_lec10.pdf).

The screenshot shows a Jupyter Notebook interface with three code cells. The first cell (In [17]) contains:

```
In [17]: li = [1,2,3]
print(li.append(5))
```

It outputs 'None'. The second cell (In [19]) contains:

```
In [19]: li = [1,2,3]
li = li.append(5)
print(li)
```

It also outputs 'None'. The third cell (In [20]) contains:

```
In [20]: li = [1,2,3]
li.append(5)
print(li)
```

It outputs '[1, 2, 3, 5]'. To the right of the notebook is a video frame of the host, and below the notebook is a large watermark 'CLASSES'.

Because of '=' symbol

# BIG IDEA

Some functions mutate the list and don't return anything.

[https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6\\_100l\\_f22\\_lec10.pdf](https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6_100l_f22_lec10.pdf)

## OPERATION ON LISTS – append

- Add an element to end of list with `L.append(element)`
- Mutates the list!

```
L = [2,1,3]
L.append(5)      → L is now [2,1,3,5]
```

`L`

Be careful! The append operation does a mutation, but returns the None object as a result.

[https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6\\_100l\\_f22\\_lec10.pdf](https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6_100l_f22_lec10.pdf)

GO CLASSES

## COMBINING LISTS

Remember strings

■ Concatenation, + operator, creates a **new** list, with copies

■ Mutate list with `L.extend(some_list)` (**copy of some\_list**)

```
L1 = [2, 1, 3]
L2 = [4, 5, 6]
L3 = L1 + L2
          → L3 is [2, 1, 3, 4, 5, 6]
```

Diagram illustrating list concatenation:

concatenation creates new list with copies

www.goclasses.in

Host

GO CLASSES

## List assignment + slicing: Demo

`a = [5, 6, 7, 8]` ← No one uses these things

`a[:2] = [3, 4]`

`a = [5, 6, 7, 8]`

`a[:3] = a[1:]`

`a = [5, 6, 7, 8]`

`a[:2] = a[1:]`

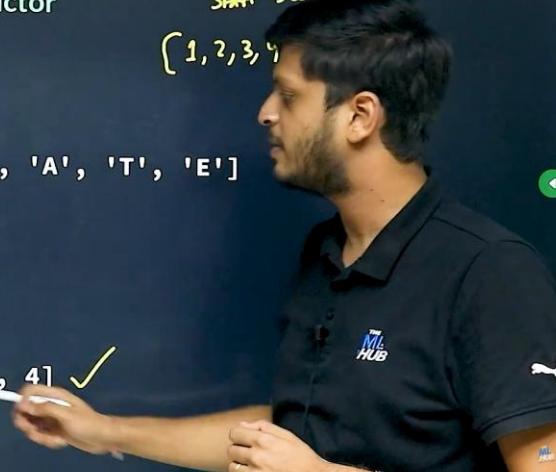
www.goclasses.in

Host

## Lists in Python

Creating Lists using `list()` constructor

```
s = "GATE"  
chars = list(s)  
print(chars)      # Output: ['G', 'A', 'T', 'E']  
  
r = range(1, 5)  
lst = list(r)  
print(lst)    # Output: [1, 2, 3, 4] ✓
```



The chalkboard has handwritten annotations for the code examples. Above the first code block, it shows `list(range(1,5))` with arrows pointing to the start (1), stop (5), and step (1). Below the second code block, it shows the output `[1, 2, 3, 4]`.

Lec 13: Lists in Python | Data Types & Control Statements in Python | Python for free | Jay Bansal

## Mathematical Operations on Lists

Operation	Example	Result
Concatenation	<code>[1, 2] + [3, 4]</code>	<code>[1, 2, 3, 4]</code>
Repetition	<code>[0] * 3</code>	<code>[0, 0, 0]</code>
Membership	<code>3 in [1, 2, 3]</code>	<code>True</code>
Length	<code>len([1, 2, 3])</code>	<code>3</code>
Min	<code>min([4, 7, 2])</code>	<code>2</code>
Max	<code>max([4, 7, 2])</code>	<code>7</code>
Sum	<code>sum([1, 2, 3])</code>	<code>6</code>



Video player controls at the bottom include play/pause, volume, and progress bar.

## List Modification Operations

insert(i, x)

Inserts an item at index i

```
lst = [1, 2, 3]
lst.insert(1, 100)
print(lst) # [1, 100, 2, 3]
```

lst = [1, 2, 3]

lst.insert(1, 100)

print(lst) # [1, 100, 2, 3]

lst = [1, 2, 3]

lst.insert(1, 100)

print(lst) # [1, 100, 2, 3]

lst = [1, 2, 3]

lst.insert(1, 100)

print(lst) # [1, 100, 2, 3]

The ML HUB

Lec 13: Lists in Python | Data Types & Control Statements in Python | Python for free | Jay Bansal

## List Modification Operations

extend(iterable)

Extends list with another list or iterable

```
lst = [1, 2, 3]
lst.extend([4, 5]) [1, 2, 3, 4, 5]
lst.extend(range(6, 8)) [1, 2, 3, 4, 5, 6, 7]
lst.extend("ABC")
print(lst) # [1, 2, 3, 4, 5, 6, 7, 'A', 'B', 'C']
```

lst = [1, 2, 3, 4, 5, 6, 7]

lst.extend("ABC")

lst.extend([1, 2, 3, 4, 5, 6, 7])

lst.extend(range(6, 8))

lst.extend("ABC")

print(lst) # [1, 2, 3, 4, 5, 6, 7, 'A', 'B', 'C']

The ML HUB

33:10 / 2:29:20

## Removing Items from List

pop([i])

Removes and returns item at index i. If index not given, removes last item.

```
lst = [1, 2, 3, 4]
print(lst.pop())      # 4
print(lst)            # [1, 2, 3]
print(lst.pop(1))     # 2
print(lst)            # [1, 3]
```

*Note: "IndexError: pop from empty list" if list is empty*

## Removing Items from List

**remove(x)**

Removes first occurrence of value x.

```
lst = [1, 2, 3, 4]
lst.remove(3)
print(lst)          # [1, 2, 4]

lst.remove(5)
# ValueError: list.remove(x): x not in list
```

## Removing Items from List

`clear()`

Removes all elements from list.

```
lst = [1, 2, 3, 4]
lst.clear()
print(lst)           # []
```



## Searching and Counting

`index(x)`

Returns index of first occurrence of value x

```
lst = [10, 20, 30]
print(lst.index(20))    # 1

print(lst.index(40))
# ValueError: 40 is not in list
```



## Searching and Counting

**count(x)**

Counts how many times x appears in the list

```
lst = [10, 20, 30]  
print(lst.count(20))
```

{20, 10, 20,

# 1

```
print(lst.count(40)) # 0
```



43:44 / 2:29:20

## Sorting and Reversing

**sort()**

Sorts list in place in ascending order

```
nums = [4, 2, 7]  
nums.sort()  
print(nums) # [2, 4, 7]  
nums.sort(reverse=True)  
print(nums) # [7, 4, 2]
```

A medium shot of the same man from the previous frame, now looking directly at the camera and gesturing with his hands while speaking. He is wearing a dark polo shirt with a small 'ML HUB' logo on the chest. He is holding a whiteboard marker in his right hand.

## Sorting and Reversing

`sorted()`  
Returns a new sorted list

```
nums = [4, 2, 7]
nums2 = sorted(nums)
print(nums2) # [2, 4, 7]
nums2 = sorted(nums, reverse=True)
print(nums2) # [7, 4, 2]
```

$a = [2, 1, 3, 4]$   
 $a.sort() \rightarrow$  changes the original list  
 $b = \underline{\text{sorted}(a)}$  → original list  $a$  remains as it is.



## Shallow and Deep Copy

Memory Behavior in Nested Lists

When we create a nested list:

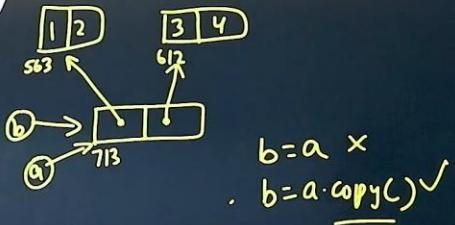
```
a = [[1, 2], [3, 4]]
```

Each inner list (like `[1, 2]` and `[3, 4]`) is a separate object in memory, and the outer list `a` contains references (pointers) to these objects.

Now, when we perform:

```
b = a
```

We're not creating a new list — just creating a new reference to the same memory locations



## Shallow and Deep Copy

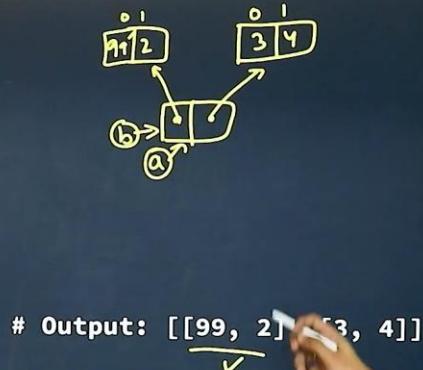
### Example

```
a = [[1, 2], [3, 4]]
```

```
b = a
```

```
b[0][0] = 99
```

```
print(a)
```



## Shallow and Deep Copy

### Shallow copy

Use list.copy() or copy.copy()

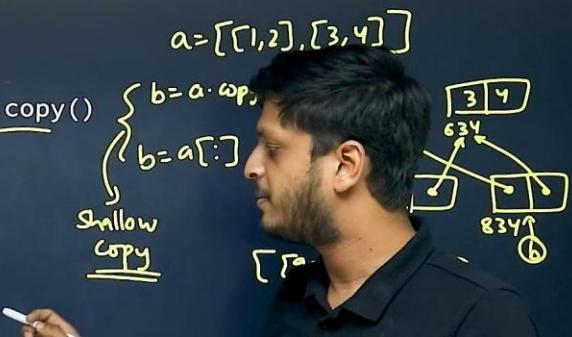
```
< - 5
```

```
a = [[1, 2], [3, 4]]
```

```
b = a.copy() ✓
```

```
b[0][0] = 99
```

```
print(a)
```



## List Comprehensions

### Examples

```
squares = [x**2 for x in range(5)]
print(squares)      # Output: [0, 1, 4, 9, 16]
```

### With Condition (Filtering):

```
even_squares = [x**2 for x in range(10) if x % 2 == 0]
print(even_squares)      # Output: [0, 4, 16, 36, 64]
```

1:18:55 / 2:29:20

**List Comprehensions**

**Examples**

$0, 1, 2, 3, 4 \rightarrow \begin{matrix} \checkmark & \checkmark & \checkmark & \checkmark \\ x & x & x & x \end{matrix}$

$\left. \begin{array}{l} \text{sq = []} \\ \text{for } x \text{ in range(10):} \\ \quad \text{if } x \% 2 == 0: \\ \quad \quad \text{sq.append(x**2)} \end{array} \right\}$

**With Condition (Filtering):**

$0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \rightarrow \begin{matrix} \checkmark & \checkmark \\ x & x & x & x & x & x & x & x & x \end{matrix}$

$\left. \begin{array}{l} \text{even_squares = [x**2 for } x \text{ in range(10) if } x \% 2 == 0] \\ \text{print(even_squares)} \end{array} \right\} \# \text{ Output: [0, 4, 16, 36, 64]}$

$\begin{matrix} \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\ 0^2 & 2^2 & 4^2 & 6^2 & 8^2 \end{matrix}$

## List Comprehensions

### Examples

↳ if...else (Conditional Expression):

```
< -5
labels = ["even" if x % 2 == 0 else "odd" for x in range(5)]
print(labels)
```

Output: ['even', 'odd', 'even', 'odd', 'even']



1:23:28 / 2:29:20

