

The screenshot shows a video call interface. In the top right corner, there is a video feed of a man wearing glasses and a white shirt, labeled "Host". The top left corner features the "GO CLASSES" logo. The main content area displays a title slide with the text "Tuples, Zip Function, and Lambda Functions". Below the title, there is a watermark for "GO CLASSES". At the bottom of the slide, there is a navigation bar with icons for volume, full screen, and other controls, along with the URL "www.goclasses.in" and a timestamp "00:22 - 1:26:26".

The screenshot shows a video call interface. In the top right corner, there is a video feed of a man wearing glasses and a white shirt, labeled "Host". The main content area contains handwritten notes on function parameters. It starts with a code snippet: 

```
def func(a, b=5, c=10):  
    print('a is = ', a, 'and b is = ', b, 'and c is = ', c)
```

 Below the code, there are three examples of function calls with annotations:

- $\text{fun}(3, 15)$  →  $a = 3$   
 $b = 15$   
 $c = 10 \}$  default
- $\text{fun}(3, \underline{\hspace{2cm}})$  → pass 'c' without specifying 'b'
- $\text{fun}(3, 5, \underline{\hspace{2cm}})$  → pass 'b' as 5 because we want it to be default

A large bracket at the bottom right groups the last two examples with the annotation "Here problem is: we need to remember default values." A timestamp at the bottom indicates the video is at 07:33 of 1:19:15.

`def func(a, b=5, c=10):  
 print('a is = ', a, 'and b is = ', b, 'and c is = ', c)`

Host

fun( 3, )  
want to pass c without passing "b"

CLASSES

08:35 - 1:18:13

`def func(a, b=5, c=10):  
 print('a is = ', a, 'and b is = ', b, 'and c is = ', c)`

Host

fun( 3, )  
want to pass c without passing "b"

fun( 3, c=24)  
so  
=

CLASSES

09:20 - 1:17:28

**GO CLASSES**



In [2]:

```
def fun(ds, ai, c=5, d=10):
    print("ds = ", ds,"ai = ", ai, "c = ", c,"d = ", d)

#my task is I want these values inside fun, ds = 10, ai = 18
fun(10,18) #I need to remember the order
```

ds = 10 ai = 18 c = 5 d = 10

Host

14:51 - 1:11:57

**GO CLASSES**



In [8]:

```
print(1,2, "GATE", 3, "DA")
```

1 2 GATE 3 DA

{ separated by space

In [9]:

```
print(1,2, "GATE", 3, "DA", sep="**")
```

1\*\*2\*\*GATE\*\*3\*\*DA

24:37 - 1:02:11

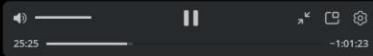


```
In [13]: print(1,2, "GATE", 3, "DA", sep="**", end="")  
print("GO", "Classes",sep="--")  
print("GATE", "Overflow",sep="$$")
```

1\*\*2\*\*GATE\*\*3\*\*DA GATE--Classes  
GATE\$\$Overflow

```
In [1]: print(1,2, "GATE", 3, "DA", end="", sep="**")  
print("GO", "Classes",sep="--")  
print("GATE", "Overflow",sep="$$")
```

1\*\*2\*\*GATE\*\*3\*\*DA GATE--Classes  
GATE\$\$Overflow



25:25 -1:01:23

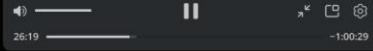


## Keyword Arguments

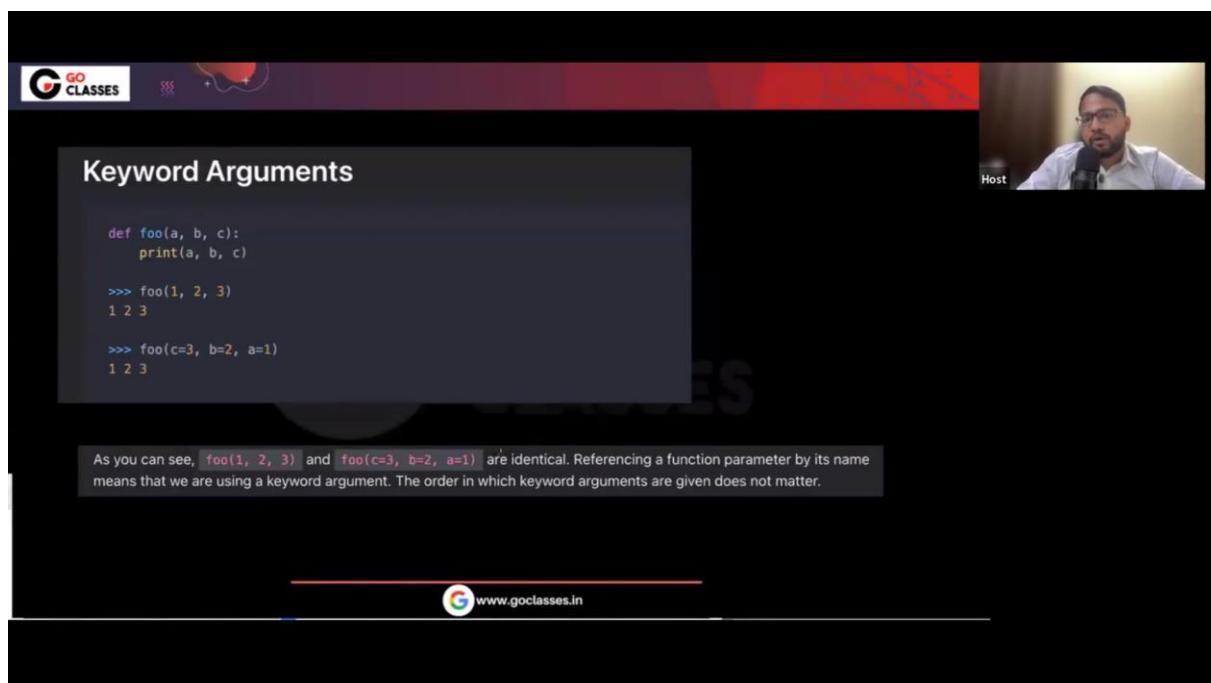
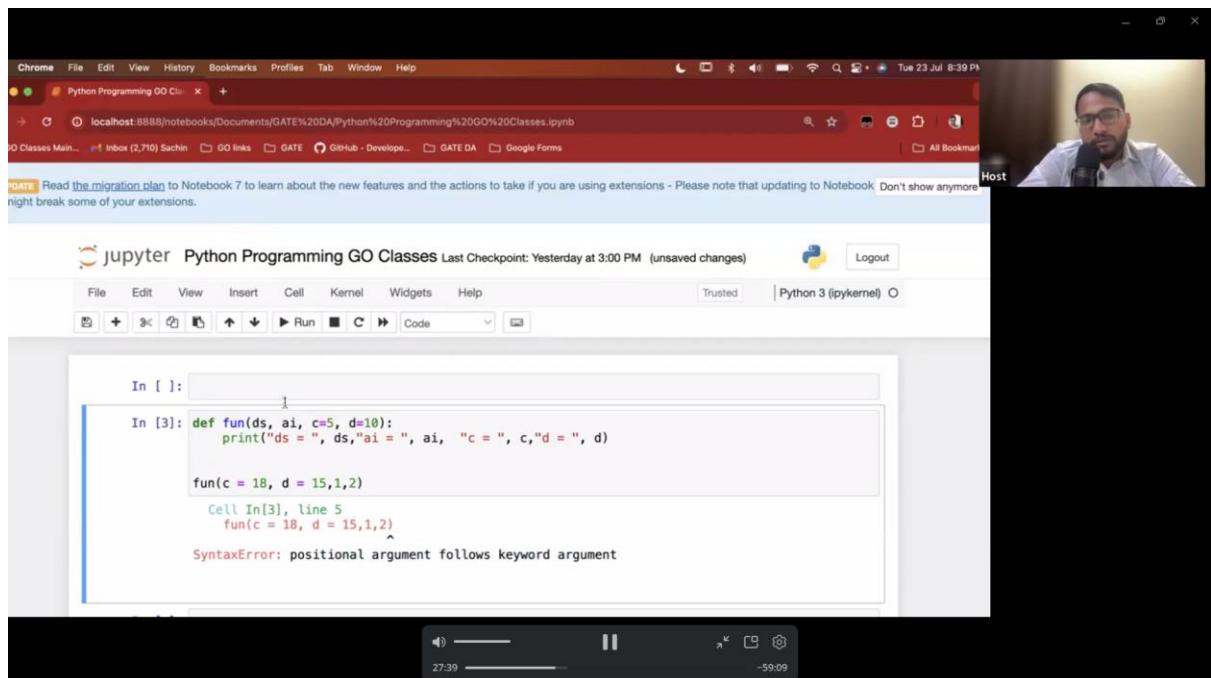
- o Functions can be called with arguments out of order
- o These arguments are specified in the call
- o Keyword arguments can be used after all other arguments.

```
>>> def myfun(a, b, c):  
    return a - b
```

```
>>> myfun(2, 1, 43)      # 1  
>>> myfun(c=43, b=1, a=2)  # 1  
>>> myfun(2, c=43, b=1) # 1  
>>> myfun(a=2, b=3, 5)  
    myfun(a=2, b=3, 5)  
                                ^  
SyntaxError: positional argument follows keyword argument
```



26:19 -1:00:29



A screenshot of a video player interface from Go Classes. The video frame shows a male host speaking into a microphone. In the bottom left corner of the video frame, the word "Host" is displayed. The main content area of the video player shows a Python interpreter session. The code shown is:

```
def foo(a=0, b=0, c=0):
    print(a, b, c)

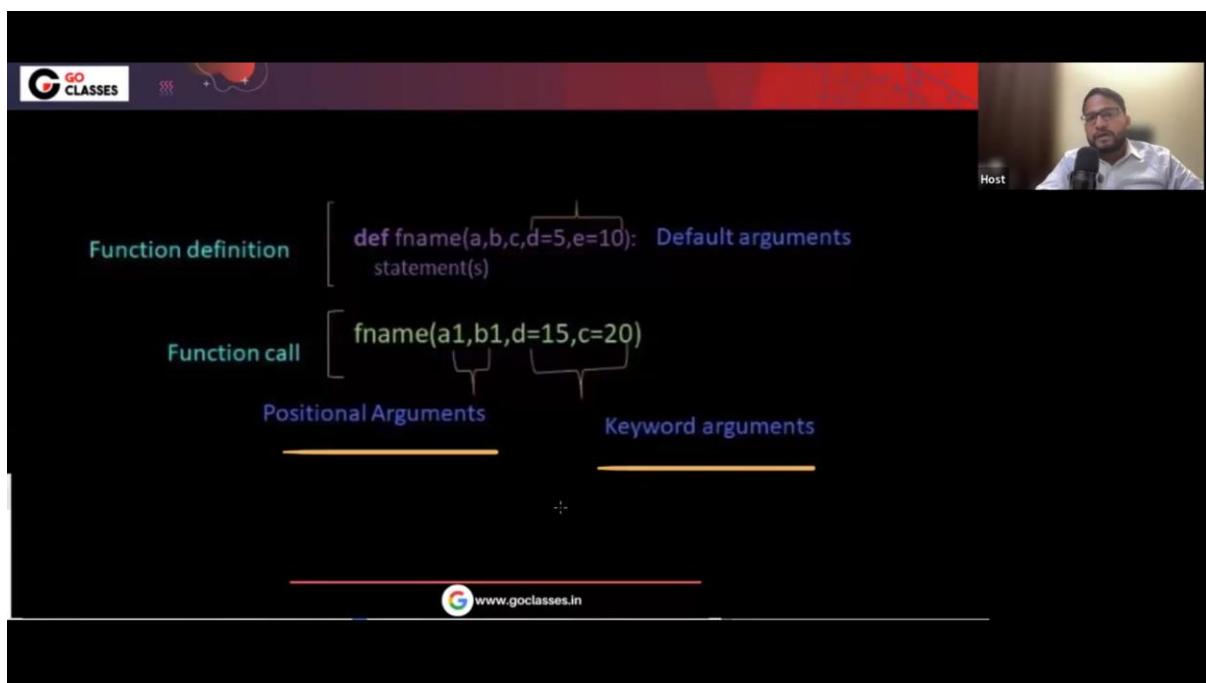
>>> foo()
0 0 0

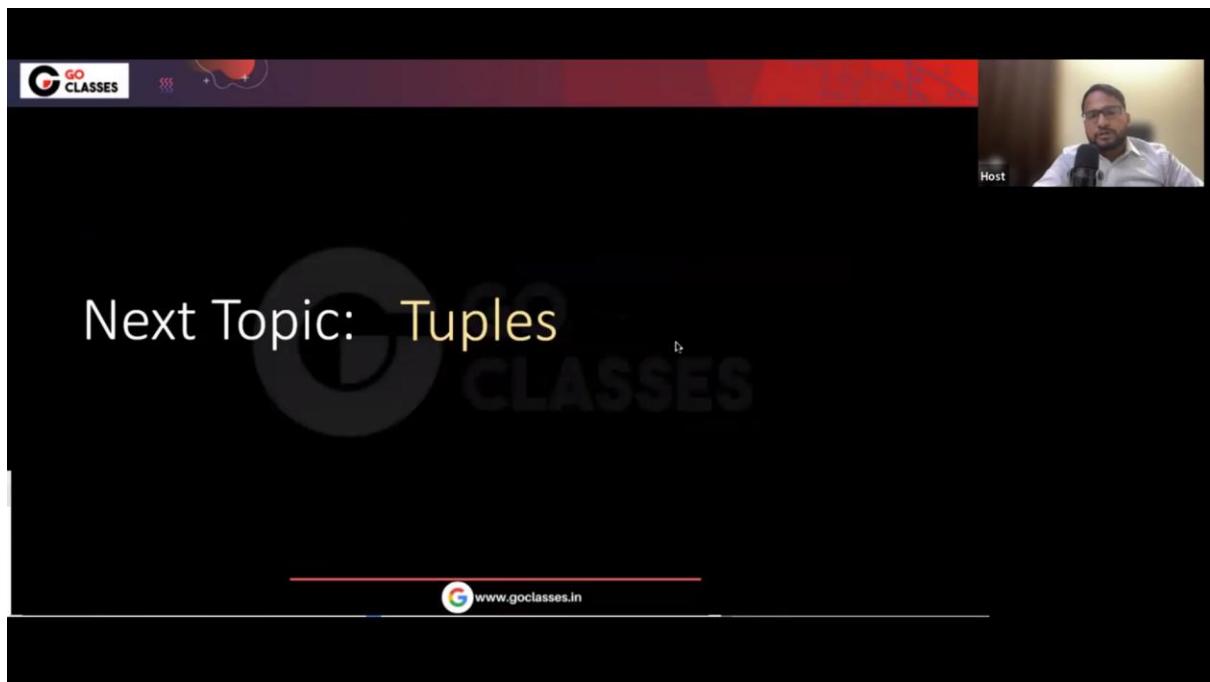
>>> foo(1, 2, 3)
1 2 3

>>> foo(c=3, b=2)
0 2 3

>>> foo(1, c=3)
1 0 3
```

The video player has a progress bar at the bottom with "30:23" on the left and "-56:25" on the right. Below the progress bar is a control bar with icons for volume, play/pause, and other media controls.





- **Tuple: an immutable sequence**
  - Very similar to a list
  - Once it is created it cannot be changed
  - Format: `tuple_name = (item1, item2)`
  - Tuples support operations as lists
    - Subscript indexing for retrieving elements
    - Methods such as `index`
    - Built in functions such as `len`, `min`, `max`
    - Slicing expressions
    - The `in`, `+`, and `*` operators

The screenshot shows a video call interface with a host on the right. The main content is a slide titled "Tuples". It lists various tuple examples and their descriptions:

( )	empty tuple
( 1, 2, 3)	integers tuple
( 1, 2.5, 3.7, 7)	numbers tuple
('a', 'b', 'c' )	characters tuple
( 'a', 1, 'b', 3.5, 'zero')	mixed values tuple
('one', 'two', 'three', 'four')	string tuple

\*Tuple is an immutable sequence whose values can not be changed.

www.goclasses.in

32:39 -54:09

The screenshot shows a video call interface with a host on the right. The main content displays Python code examples for tuples:

```
tupleEmpty = () #Empty Tuple
```

```
tupleNum = (1, 2, 3) #Tuple with Integers
```

```
tupleString = ("apple", "banana", "cherry") #Tuple with Strings
```

```
tupleMix = (1, "Hello", 3.4) #Tuple with Mixed Data Types
```

www.goclasses.in

33:16 -53:32

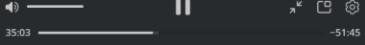


```
In [7]: a = (1,2,3)
print(a, type(a))

b = [1,2,3]
print(b,type(b))

c = tuple(b)          #convert list to tuple
print(c,type(c))

(1, 2, 3) <class 'tuple'>
[1, 2, 3] <class 'list'>
(1, 2, 3) <class 'tuple'>
```

A video player interface showing playback controls: volume, progress bar, and timestamp.

The image shows a screenshot of a video player interface for 'GO CLASSES'. At the top left is the channel logo. On the right, there is a video feed of a man with glasses and a beard, labeled 'Host'. Below the video feed is a control bar with a play button, volume slider, and timestamp markers '35:43' and '-51:05'. The main content area displays two code snippets in a Jupyter-style notebook interface.

**In [9]:**

```
a = (1,2,3)
print(a, type(a))

b = list(a)
print(b, type(b))
```

(1, 2, 3) <class 'tuple'>
[1, 2, 3] <class 'list'>

**In [10]:**

```
a = (10)

print(a, type(a))
```

10 <class 'int'>

**In [11]:**

```
a = (10,)

print(a, type(a))
```

(10,) <class 'tuple'>

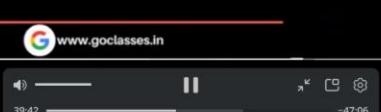
**Tuple unpacking**

```
>>> val = (10,20,30)
>>> a,b,c = val
```

```
In [12]: a = (10,20,30)
b = a
b
Out[12]: (10, 20, 30)
```

```
In [13]: a = (10,20,30)
num1, num2, num3 = a
print(num1, num2, num3)
10 20 30
```

Host



**Parentheses () are optional**

```
In [1]: x = 5,4
x
Out[1]: (5, 4)
```

Host



The screenshot shows a video player interface for a Go Classes lesson. The title "Tuple – Can have mixed data types" is displayed in blue text. A small video window in the top right corner shows a male host speaking. Below the title, a bulleted list states: "• A tuple can have any number of items and they may be of different types (integer, float, [list], "string" etc.)". Below the list is a code snippet: `tupleMix = ("mouse", [8, 4, 6], (1, 2, 3))`. The URL [www.goclasses.in](http://www.goclasses.in) is visible at the bottom of the screen.

The screenshot shows a video player interface for a Go Classes lesson. The title "Access Items in a Tuple" is displayed in blue text. A small video window in the top right corner shows a male host speaking. Below the title, there is sample code: `fruitTup = ("apple", "banana", "cherry")`, followed by three print statements: `print(fruitTup[0])`, `print(fruitTup[1])`, and `print(fruitTup[2])`. To the right of the code, a table illustrates tuple indexing:

	0	1	2
fruitTup	fruitTup[0] apple	fruitTup[1] banana	fruitTup[2] cherry

The video player controls at the bottom include a progress bar from 41:34 to ~45:14, a play/pause button, and other standard media controls.

A screenshot of a video lesson from Go Classes. The host is visible in the top right. The main content area shows handwritten notes comparing lists and tuples. On the left, a list is shown:

```
t = [2, "mit", 3]
```

An annotation points to the assignment of 4 to index 1 with the text: "t[1] = 4 → gives error, can't modify object". Below this, handwritten text reads: "this is the difference between list and tuple".

On the right, a string is shown:

```
s = "Python"
```

An annotation points to s[0] with the text: "s[0] = 'P' {error}". Below this, handwritten text reads: "S = "program"" and "this is good".

At the bottom of the slide, there is a link: [https://ocw.mit.edu/courses/6-100L-introduction-to-cs-and-programming-using-python-fall-2022/mit6\\_100L\\_f22\\_lec09.pdf](https://ocw.mit.edu/courses/6-100L-introduction-to-cs-and-programming-using-python-fall-2022/mit6_100L_f22_lec09.pdf).

A screenshot of a video lesson from Go Classes. The host is visible in the top right. The main content area shows handwritten notes on indices and slicing. At the top, it says "INDICES AND SLICING" and "Remember strings?".

Below, a sequence of elements is defined:

```
seq = (2, 'a', 4, (1,2))
```

The indices are listed as 0, 1, 2, 3. To the right, a series of print statements and their outputs are shown:

print(len(seq))	→ 4	}
print(seq[3])	→ (1,2)	
print(seq[-1])	→ (1,2)	
print(seq[3][0])	→ 1	
print(seq[4])	→ error	

Handwritten notes next to this group say: "An element of a sequence is at an index, indices start at 0".

print(seq[1])	→ 'a'	}
print(seq[-2:])	→ (4, (1,2))	
print(seq[1:4:2])	→ ('a', (1,2))	
print(seq[:-1])	→ (2, 'a', 4)	
print(seq[1:3])	→ ('a', 4)	

Handwritten notes next to this group say: "Slices extract subsequences. Indices evaluated from left to right".

At the bottom, there is a watermark: www.goclasses.in

Host

- Conveniently used to swap variable values

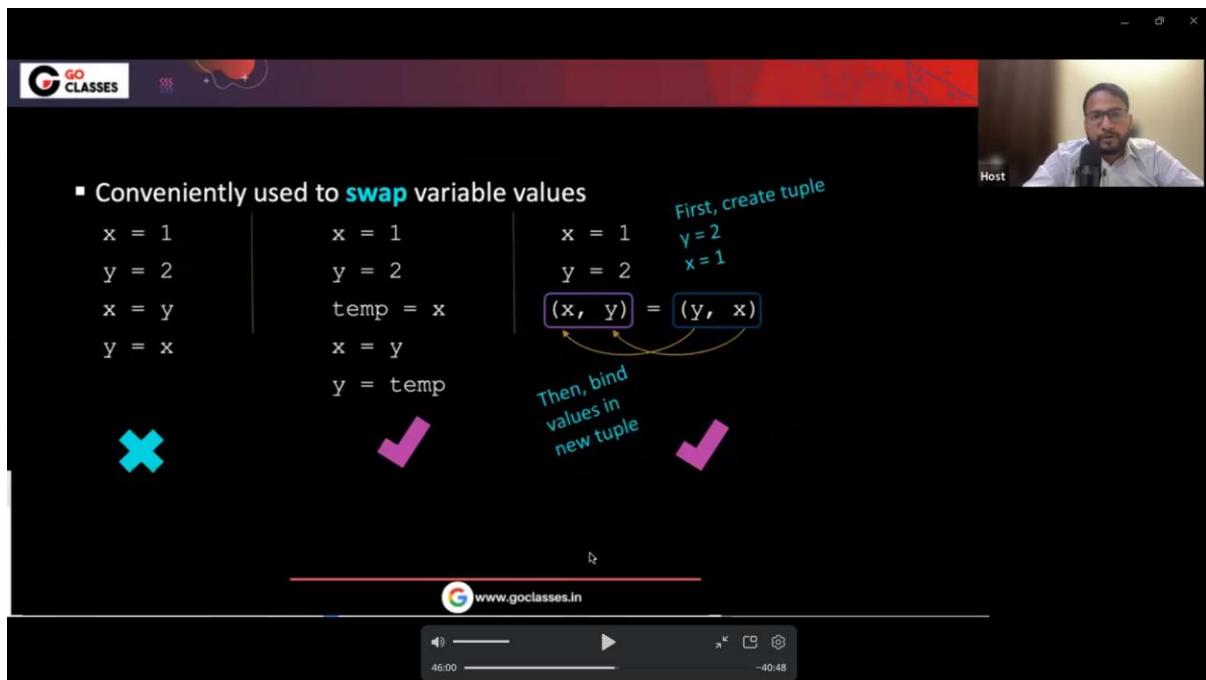
<pre>x = 1 y = 2 x = y y = x</pre>	<pre>x = 1 y = 2 temp = x x = y y = temp</pre>	<pre>x = 1 y = 2 x = 1 (x, y) = (y, x)</pre>
--	--	--

**X**      **✓**      **✓**

First, create tuple  
Then, bind values in new tuple

www.goclasses.in

46:00 -40:48



Host

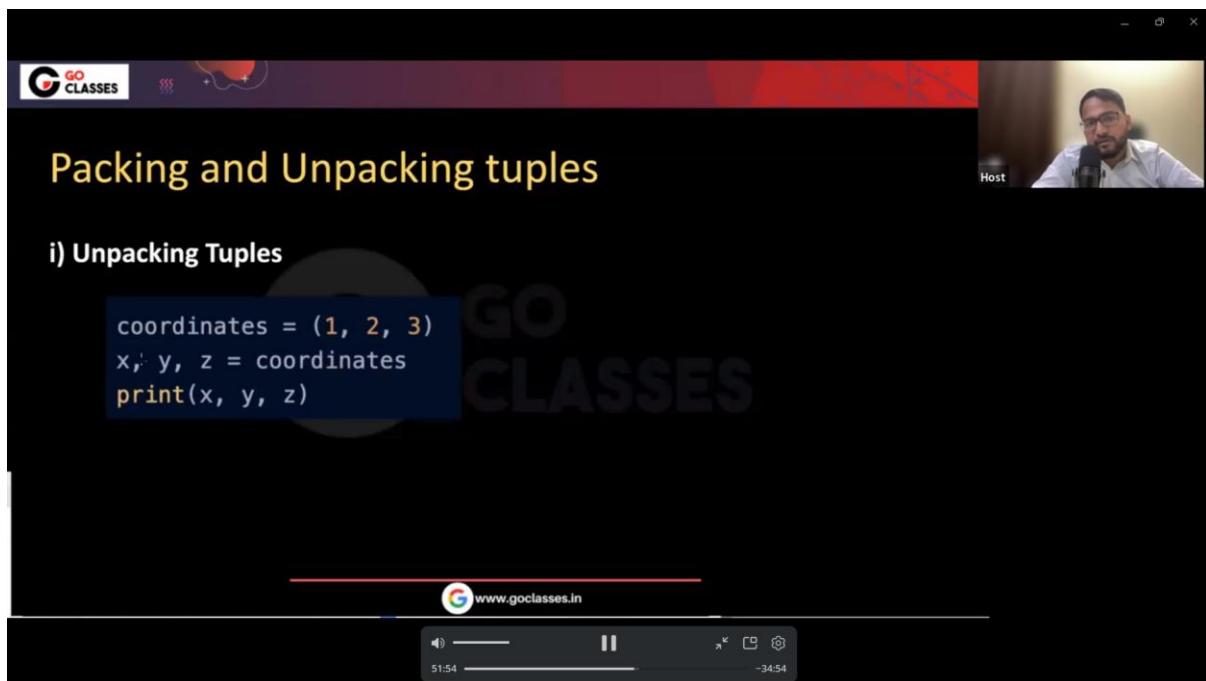
## Packing and Unpacking tuples

### i) Unpacking Tuples

```
coordinates = (1, 2, 3)  
x, y, z = coordinates  
print(x, y, z)
```

www.goclasses.in

51:54 -34:54



A screenshot of a Go Classes video session. On the left, there is a dark background with a large, semi-transparent watermark of the 'G CLASSES' logo. On the right, a video player window shows a male host with glasses and a light blue shirt. The video player interface includes a play button, volume controls, and a progress bar from 53:24 to -33:24. In the center, a white code editor window displays the following Python code:

```
# Nested tuple
nested_tuple = (1, (2, 3))

# Unpacking
a, (b, c) = nested_tuple
```

In [22]: t = 1,(2,3)  
a,b = t  
print(a, b)  
1 (2, 3)

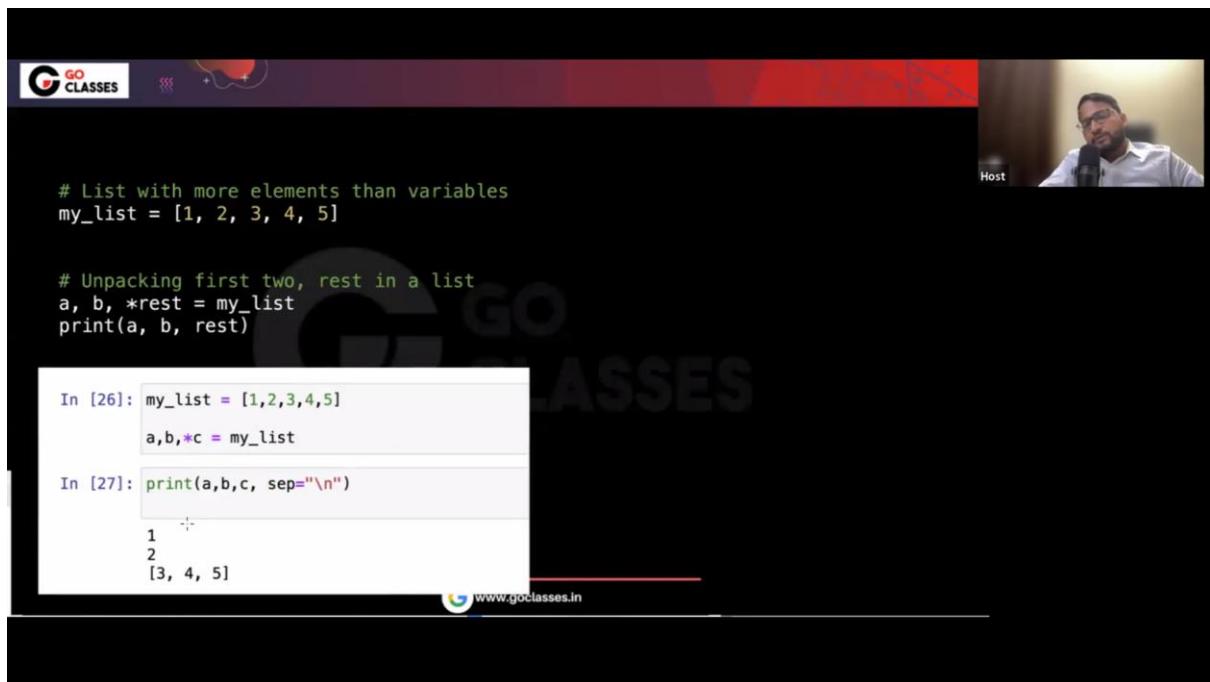
In [23]: t = 1,(2,3)  
a,(b,c) = t  
print(a, b,c)  
1 2 3

A screenshot of a Go Classes video session. On the left, there is a dark background with a large, semi-transparent watermark of the 'G CLASSES' logo. On the right, a video player window shows the same male host. The video player interface includes a play button, volume controls, and a progress bar from 53:24 to -33:24. In the center, a white code editor window displays the following Python code:

```
# String of characters
my_string = "abc"

# Unpacking
x, y, z = my_string
print(x, y, z)
```

In [24]: a,b,c = "xyz"  
print(a, b,c)  
x y z



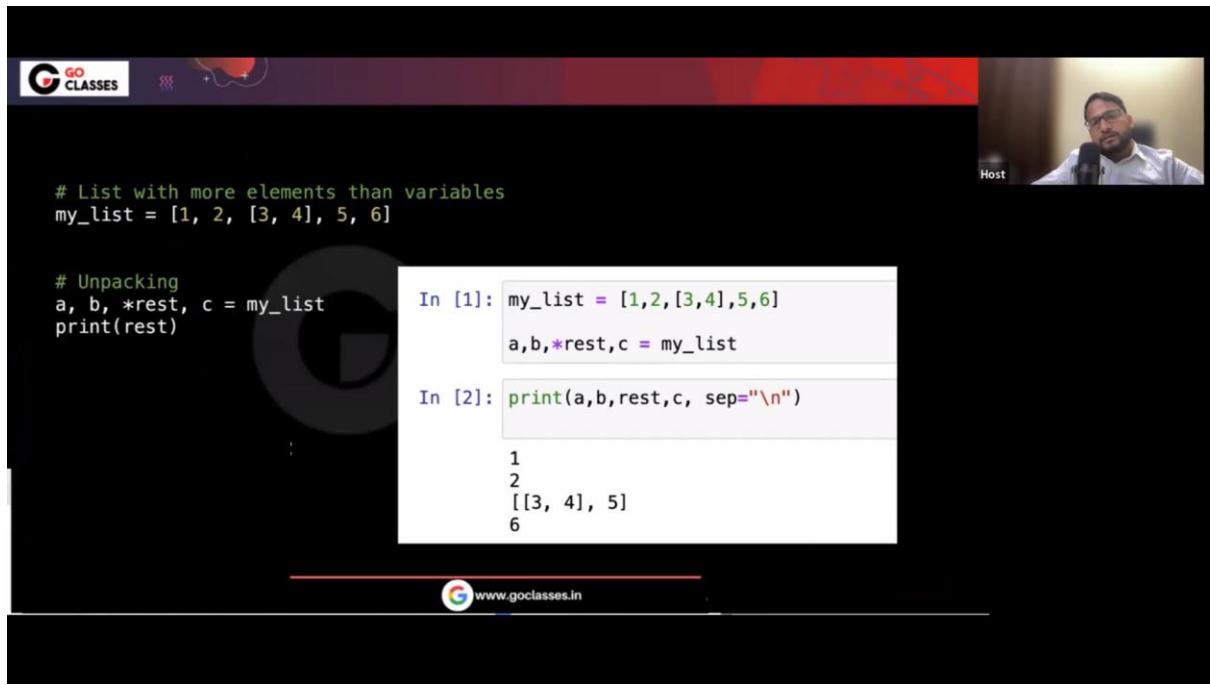
The screenshot shows a video player interface for 'GO CLASSES'. In the top right corner, there is a video feed of a man with glasses and a white shirt, labeled 'Host'. The main area displays a Jupyter Notebook code cell. The code consists of two lines:

```
# List with more elements than variables  
my_list = [1, 2, 3, 4, 5]  
  
# Unpacking first two, rest in a list  
a, b, *rest = my_list  
print(a, b, rest)
```

Below the code cell, the output of the code execution is shown in two lines:

```
In [26]: my_list = [1,2,3,4,5]  
a,b,*c = my_list  
  
In [27]: print(a,b,c, sep="\n")  
1  
2  
[3, 4, 5]
```

A watermark for 'www.goclasses.in' is visible at the bottom center of the screen.



The screenshot shows a video player interface for 'GO CLASSES'. In the top right corner, there is a video feed of a man with glasses and a white shirt, labeled 'Host'. The main area displays a Jupyter Notebook code cell. The code consists of three lines:

```
# List with more elements than variables  
my_list = [1, 2, [3, 4], 5, 6]  
  
# Unpacking  
a, b, *rest, c = my_list  
print(rest)
```

Below the code cell, the output of the code execution is shown in three lines:

```
In [1]: my_list = [1,2,[3,4],5,6]  
a,b,*rest,c = my_list  
  
In [2]: print(a,b,rest,c, sep="\n")  
1  
2  
[[3, 4], 5]  
6
```

A watermark for 'www.goclasses.in' is visible at the bottom center of the screen.

GO CLASSES

Host

```
In [1]: my_list = [1,2,5,6]
a,b,*c,d = my_list

In [2]: print(a,b,c,d, sep="\n")
1
2
[5]
6
```

```
In [3]: my_list = [1,2,5,6]
a,b,c,d = my_list

In [4]: print(a,b,c,d, sep="\n")
1
2
5
6
```

GO CLASSES

Host

```
# List with more elements than variables
my_list = [1, 2, 3, 4]

# Unpacking
a, b, c = my_list

print(a,b,c)
```

L

```
In [5]: my_list = [1,2,3,4]
a,b,c = my_list
print(a,b,c)

ValueError                                Traceback (most recent call last)
Cell In[5], line 3
      1 my_list = [1,2,3,4]
      2
----> 3 a,b,c = my_list
      4
      5 print(a,b,c)

ValueError: too many values to unpack (expected 3)
```

www.goclasses.in

The screenshot shows a video player interface for Go Classes. In the top right corner, there is a video feed of a host wearing glasses and a white shirt. The video title is "Python Unpacking". The main content area displays a Jupyter Notebook cell with the following code:

```
# List with more elements than variables
my_list = [1, 2, 3, 4]

# Unpacking
a, b, c = my_list

print(a,b,c)
```

Below this, another cell shows:

```
In [5]: my_list = [1,2,3,4]
a,b,c = my_list
print(a,b,c)

ValueError
Cell In[5], line 3
  1 my_list = [1,2,3,4]
  ----> 3 a,b,c = my_list
      5 print(a,b,c)

ValueError: too many values to unpack (expected 3)
```

On the left side of the screen, there is a sidebar with the Go Classes logo and some decorative icons.

This screenshot continues the video from the previous one. The host is still visible in the top right. The notebook cells show the same code as before, but the error message has changed:

```
# List with more elements than variables
my_list = [1, 2, 3, 4]

# Unpacking
a, b, c, d, e = my_list

print(a,b,c,d,e)
```

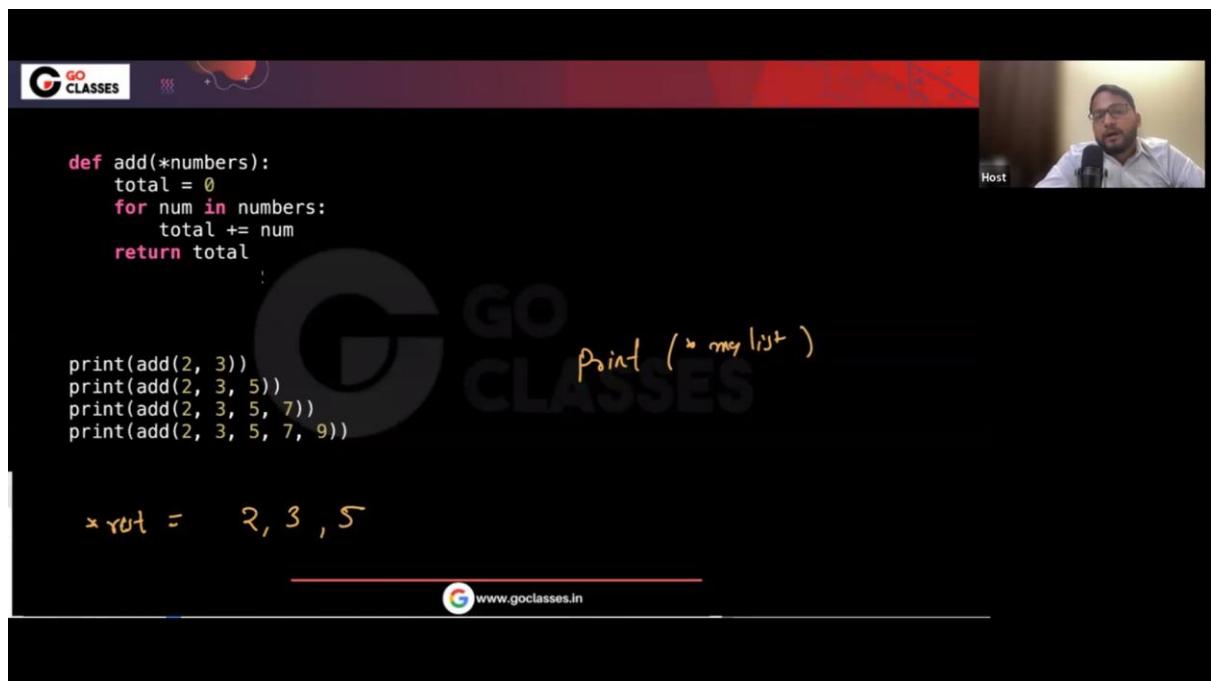
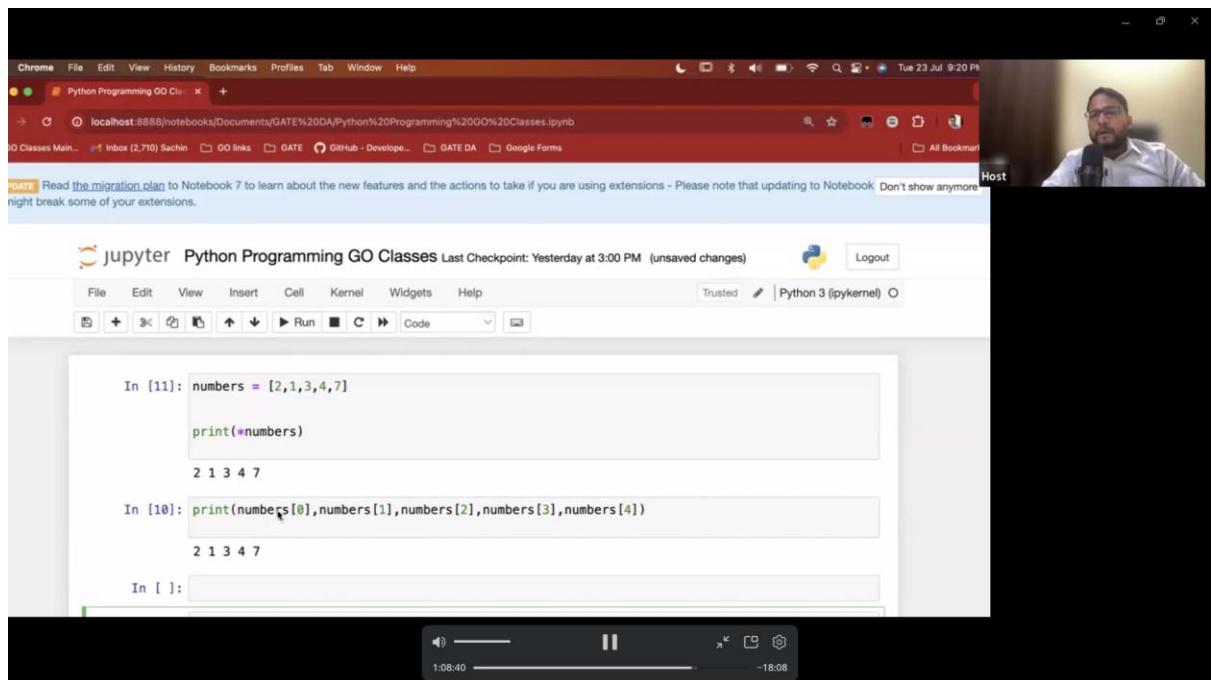
Below this, another cell shows:

```
In [7]: my_list = [1,2,3,4]
a,b,c,d,e = my_list
print(a,b,c,d,e)

ValueError
Cell In[7], line 3
  1 my_list = [1,2,3,4]
  ----> 3 a,b,c,d,e = my_list
      5 print(a,b,c,d,e)

ValueError: not enough values to unpack (expected 5, got 4)
```

The video player interface includes a progress bar at the bottom indicating the video is at 1:06:11 of 2:17 minutes.



**GO CLASSES**

In [8]:

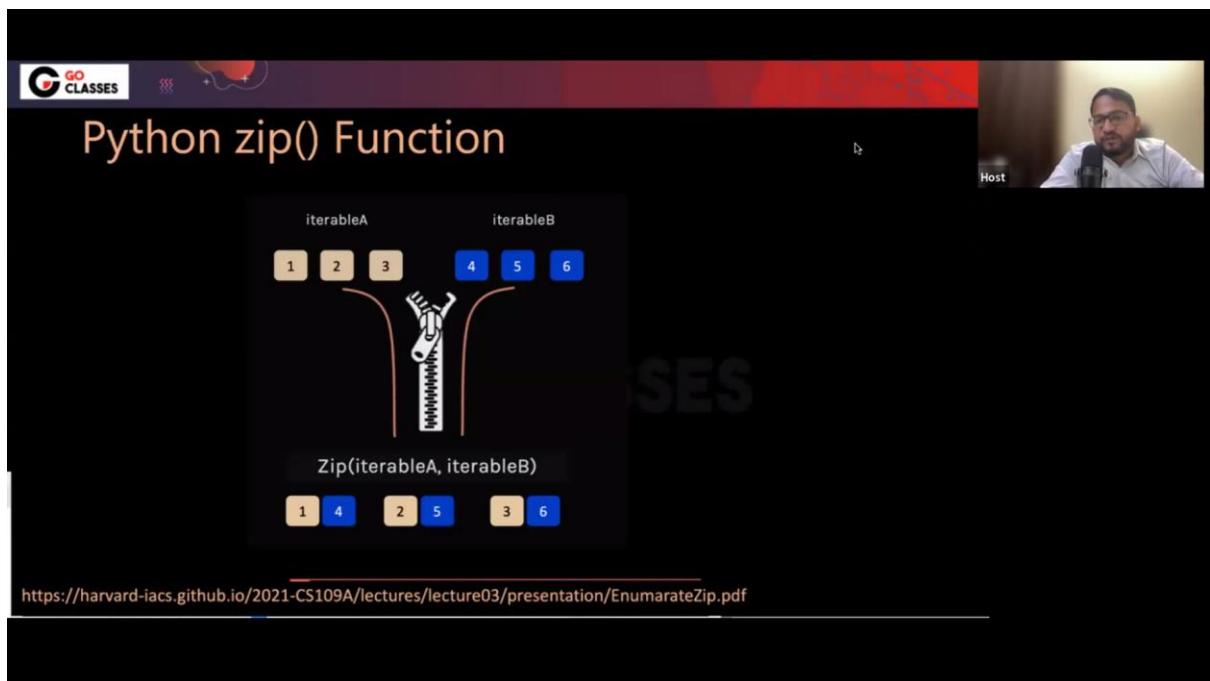
```
def fun(*numbers):
    s = 0
    for num in numbers:
        s+=num
    print(s)

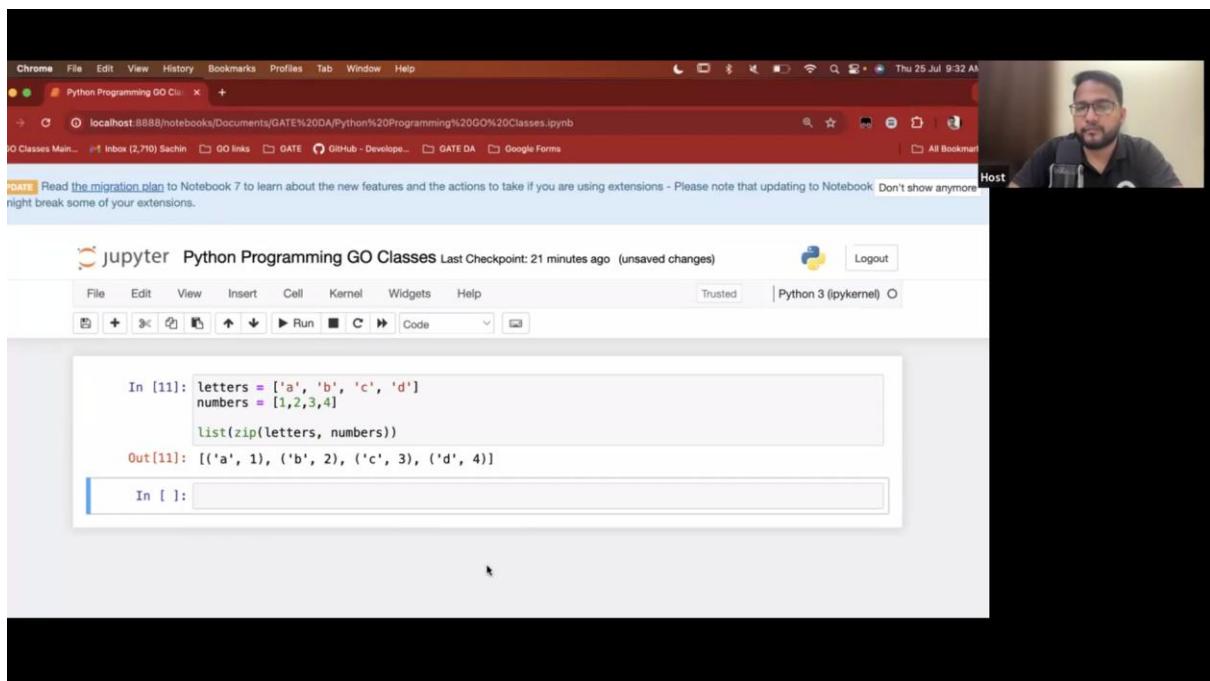
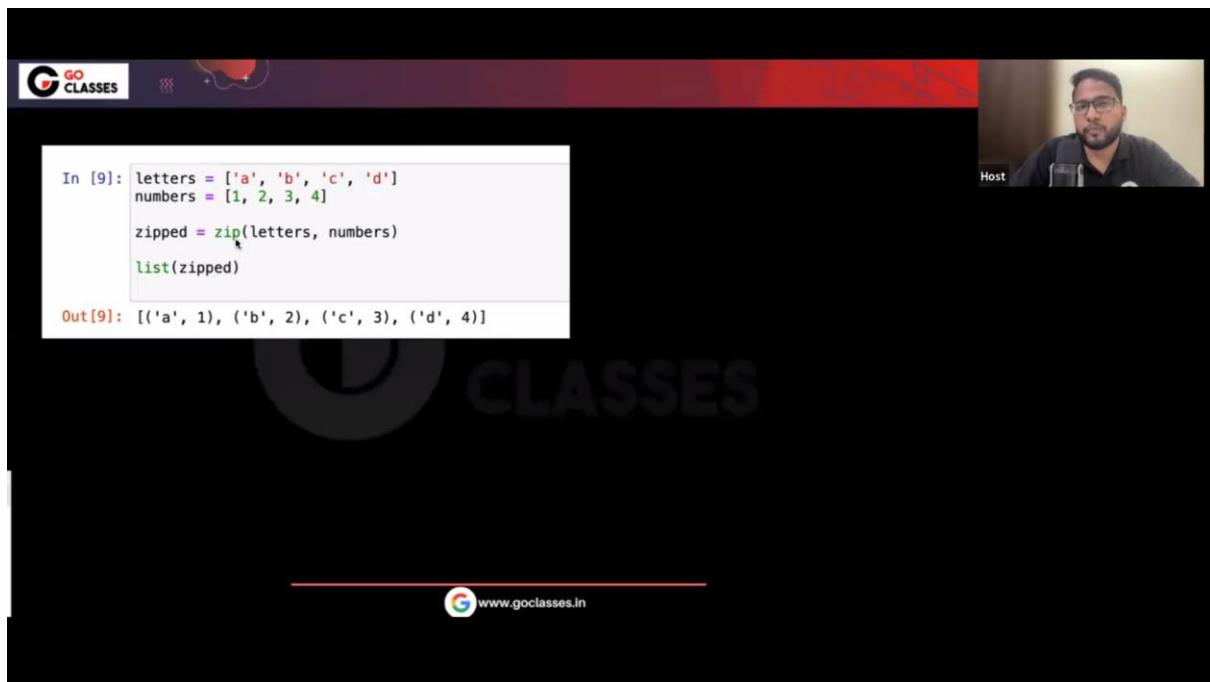
fun(1,2,3)
fun(1,2,3,4)
fun(1,2,3,4,5)
```

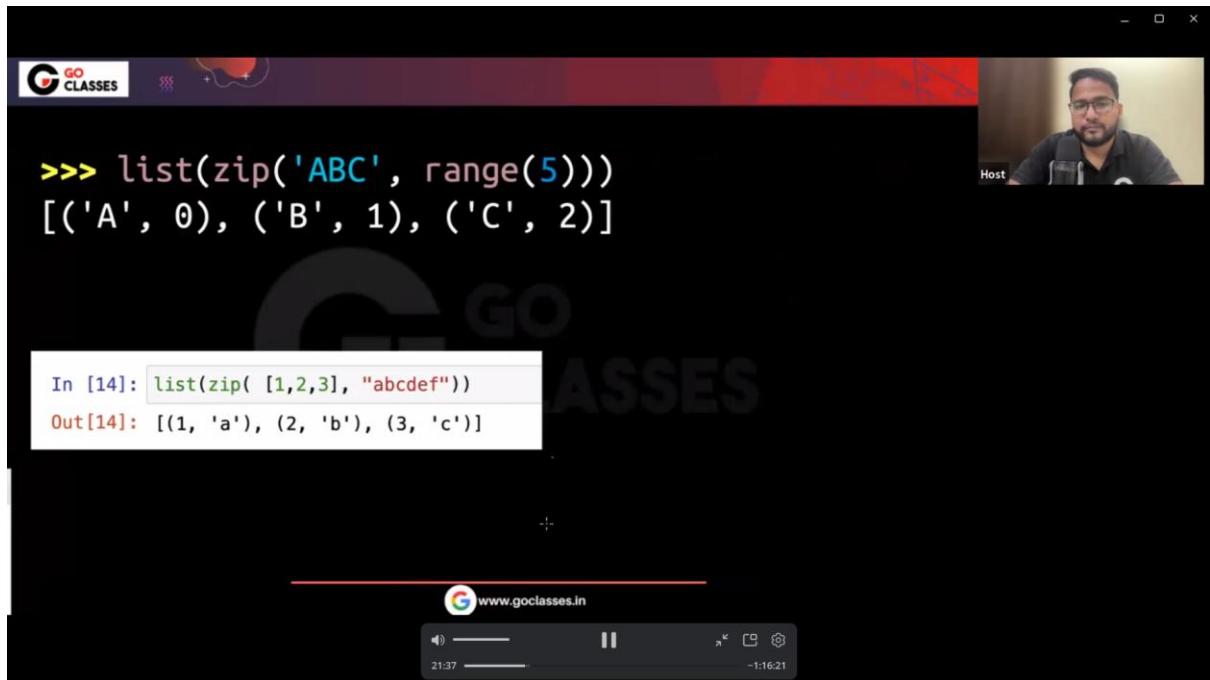
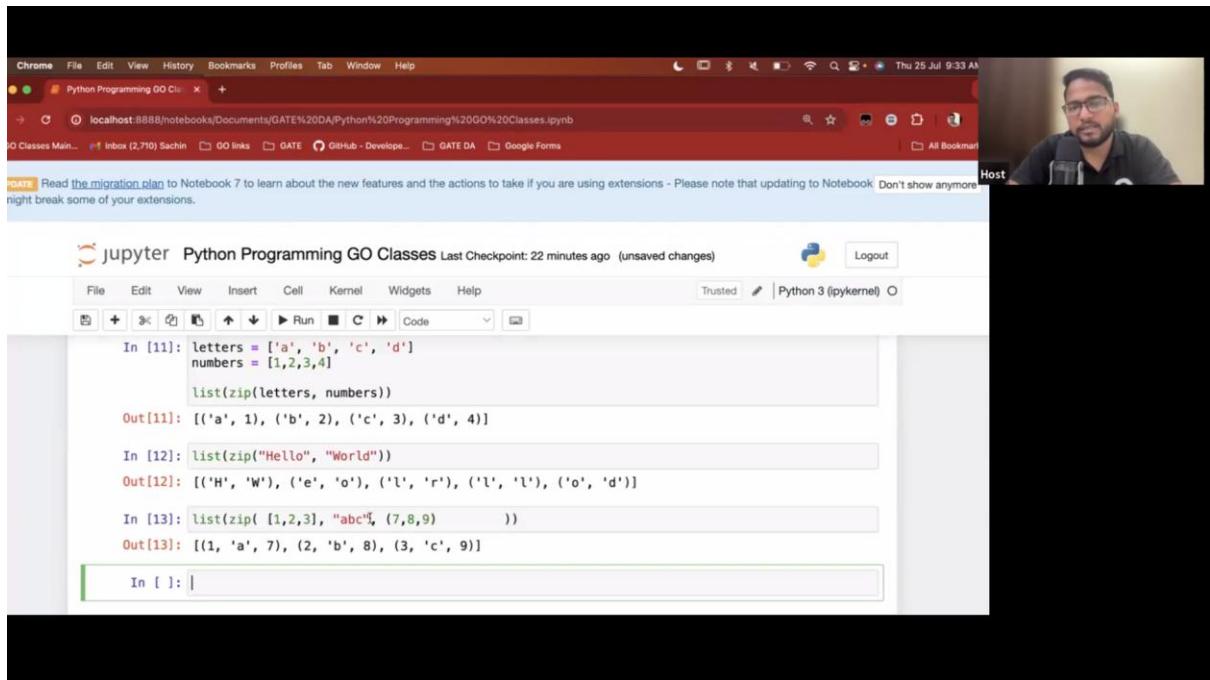
6  
10  
15

Host

1:19:02 -> 07:46







A screenshot of a video player interface for 'GO CLASSES'. The video frame shows a man with glasses and a dark shirt, identified as the 'Host'. The video player has a red and black decorative bar at the top. In the bottom right corner of the video frame, there is a small 'Host' label with a camera icon. The video content itself is a Python terminal session:

```
>>> list(zip('ABC', range(5), [10, 20, 30, 40]))  
[('A', 0, 10), ('B', 1, 20), ('C', 2, 30)]
```

The video player interface includes a play/pause button, volume controls, and a progress bar indicating the video is at 21:43 of a total duration of 1:16:15.

A screenshot of a video player interface for 'GO CLASSES'. The video frame shows the same host as before. A callout box contains a note about the 'zip()' function:

- By default, `zip()` stops when the shortest iterable is exhausted. It will ignore the remaining items in the longer iterables, cutting off the result to the length of the shortest iterable:

```
>>> list(zip(range(3), ['fee', 'fi', 'fo', 'fum']))  
[(0, 'fee'), (1, 'fi'), (2, 'fo')]
```

The video player interface includes a play/pause button, volume controls, and a progress bar indicating the video is at 21:43 of a total duration of 1:16:15.

At the bottom of the screen, a URL is displayed: <https://docs.python.org/3/library/functions.html#zip>

Chrome File Edit View History Bookmarks Profiles Tab Window Help

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

Host

In [17]: `for t in zip("abc", range(1,5)):  
 print(t)`

Out[17]: `('a', 1)  
(‘b’, 2)  
(‘c’, 3)`

In [19]: `list(zip("abc", range(1,5)))`

Out[19]: `[('a', 1), ('b', 2), ('c', 3)]`

In [ ]:

24:51 -1:13:07

Chrome File Edit View History Bookmarks Profiles Tab Window Help

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

Host

In [17]: `for t in zip("abc", range(1,5)):  
 print(t)`

Out[17]: `('a', 1)  
(‘b’, 2)  
(‘c’, 3)`

In [2]: `x, *rest = zip("abc", range(1,5)) #forcing the evaluation  
print(rest)`

Out[2]: `[('b', 2), ('c', 3)]`

In [3]: `zipped_object = zip("abc", range(1,5)) #NOT forcing the evaluation  
print(zipped_object)`

Out[3]: `<zip object at 0x105035640>`

In [ ]:

28:00 -1:09:58

The screenshot shows a video player interface for Go Classes. On the left, there is a code editor window displaying three examples of the `zip` function:

```
1 for tup in zip(['a','b','c'],[1,2,3,4]):  
2     print(tup)  
('a', 1)  
('b', 2)  
('c', 3)  
  
1 for tup in zip(['a','b','c','d'],[1,2,3]):  
2     print(tup)  
('a', 1)  
('b', 2)  
('c', 3)  
  
1 for tup in zip([1,2,3],['a','b','c'],'xyz'):  
2     print(tup)  
(1, 'a', 'x')  
(2, 'b', 'y')  
(3, 'c', 'z')
```

Two callout boxes with arrows point from the text to specific parts of the code:

- A box labeled "Given arguments of different lengths, zip defaults to the shortest one." points to the first example where `zip` takes two lists of different lengths.
- A box labeled "zip takes any number of arguments, so long as they are all iterable. Sequences are iterable." points to the third example where `zip` is used with a list, a tuple, and a string.

On the right side of the video player, there is a video feed of a host wearing glasses and a dark shirt. The word "Host" is displayed above the video feed. At the bottom center of the screen is the website URL [www.goclasses.in](http://www.goclasses.in).

The screenshot shows a Jupyter Notebook interface running in a browser window. The title bar indicates it's a "Python Programming GO Classes" notebook. The code cell contains the following Python code:

```
In [3]: x = [10, 20, 30]  
y = [7, 5, 3]  
  
pairs = list(zip(x, y))  
  
print(pairs)  
  
new_x,new_y = list(zip(pairs[0], pairs[1], pairs[2]))  
  
print(new_x)  
print(new_y)  
  
[(10, 7), (20, 5), (30, 3)]  
(10, 20, 30)  
(7, 5, 3)
```

Below the code cell, another cell is partially visible with the text "In []: for t in zip("abc", range(1,5)):" followed by three dots. The bottom of the screen shows a toolbar with volume, play/pause, and other controls, along with a timestamp of "36:45" and "-1:01:13".

Chrome File Edit View History Bookmarks Profiles Tab Window Help

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

Host

```
jupyter Python Programming GO Classes Last Checkpoint: 42 minutes ago (unsaved changes)
```

In [6]:

```
x = [10, 20, 30]
y = [7, 5, 3]

pairs = list(zip(x, y))

#print(pairs)

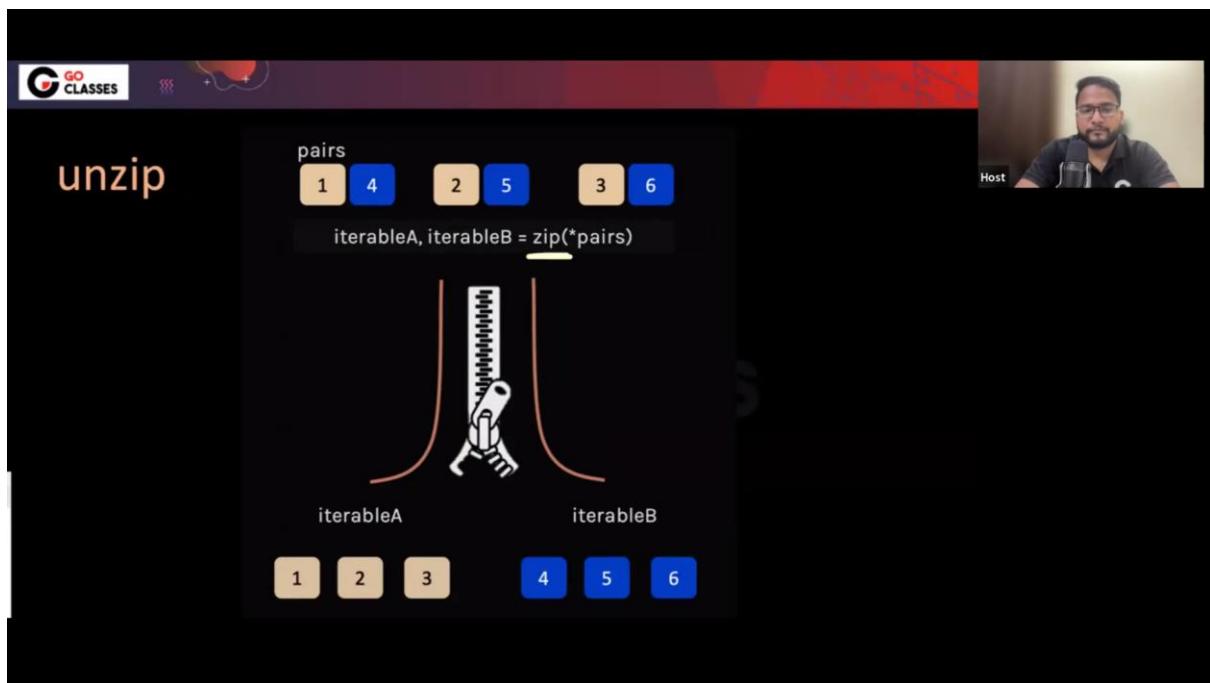
#new_x,new_y = list(zip(pairs[0], pairs[1], pairs[2]))

new_x,new_y = list(zip(*pairs))      #shortcut to above line

print(new_x)
print(new_y)

(10, 20, 30)
(7, 5, 3)
```

38:33 -59:25



A screenshot of a video player interface for 'GO CLASSES'. The video frame shows a male host with glasses speaking into a microphone. The video player controls at the bottom indicate the video is at 40:34 of a 57:24 duration. The main content area displays Python code in a terminal window:

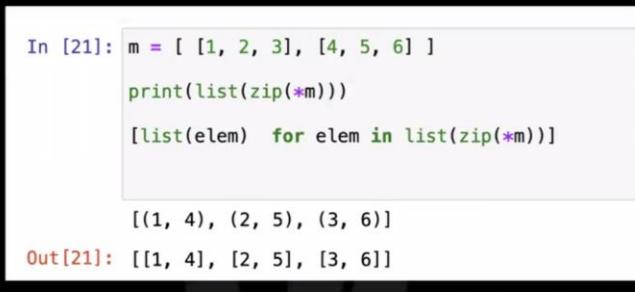
```
>>> pairs =  
[(1,'a'),(2,'b'),(3,'c')]  
>>> numbers, letters = zip(*pairs)  
  
>>> print(numbers)  
(1,2,3)  
  
>>> print(letters)  
('a','b','c')
```

A screenshot of a video player interface for 'GO CLASSES'. The video frame shows the same male host. The video player controls at the bottom indicate the video is at 40:34 of a 57:24 duration. The main content area displays Python code with annotations:

```
parts = ([['1', '0'], ['5', '5'], [['1', '9'], ['1', '1']]])  
parts = list(zip(*parts)) → [ [1,0] , [5,5] ] , [ [1,9] , [1,1] ]  
# Printing the zipped parts  
print("Zipped parts[0]:", parts[0])
```

Annotations include a yellow bracket under the first two elements of the list, and a yellow arrow pointing from the word 'zip' to the first element of the list.

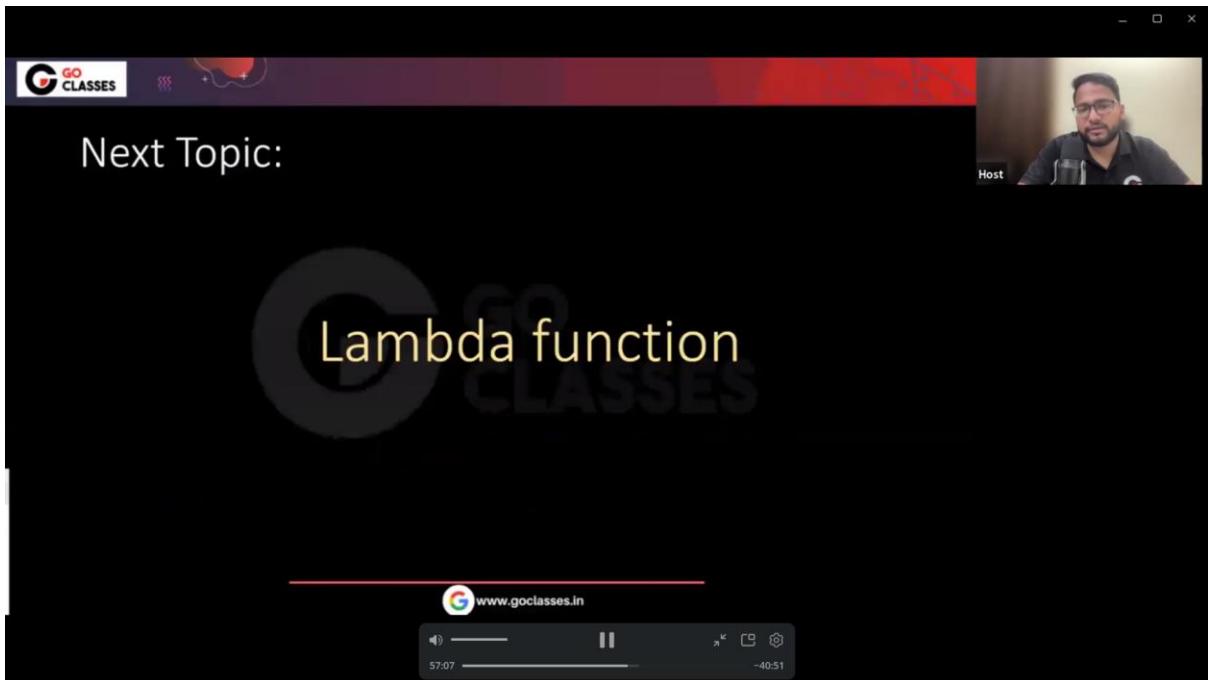
www.goclasses.in



In [21]: `m = [[1, 2, 3], [4, 5, 6]]  
print(list(zip(*m)))  
[list(elem) for elem in list(zip(*m))]  
  
[(1, 4), (2, 5), (3, 6)]`

Out[21]: `[[1, 4], [2, 5], [3, 6]]`

$m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \xrightarrow{\text{transpose}} \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$



Lambda functions are special functions defined using the following syntax:

```
lambda parameters: expression
```

lambda n: n \* 2

lambda x, y: x \*\* y

↑                   ↑  
  parameter(s)      expression

[https://web.stanford.edu/class/archive/cs/cs106ap/cs106ap.1198/lectures/25-LambdasCustomSort/25-Lambdas\\_Custom\\_Sort.pdf](https://web.stanford.edu/class/archive/cs/cs106ap/cs106ap.1198/lectures/25-LambdasCustomSort/25-Lambdas_Custom_Sort.pdf)

EXAMPLE:

```
add = lambda x, y: x + y  
add(3,5) #Output:
```

[www.goclasses.in](http://www.goclasses.in)

The screenshot shows a video player interface for 'GO CLASSES'. In the top right corner, there is a video feed of a host wearing glasses and a dark shirt. The video player has standard controls for volume, play/pause, and progress (1:00:15 to -37:43). The main content area displays a Jupyter Notebook cell. The code in the cell is:

```
In [23]: fun = lambda x: x**2
          fun(4)
Out[23]: 16
```

Handwritten annotations on the screen include:

- A yellow checkmark icon is placed next to the word "Single Expression".
- A yellow arrow points from the handwritten word "input" to the parameter "x" in the lambda function definition.
- A yellow arrow points from the handwritten text "no return statement needed" to the absence of a return statement in the code.

- 
- The screenshot shows a video player interface for 'GO CLASSES'. In the top right corner, there is a video feed of a host wearing glasses and a dark shirt. The video player has standard controls for volume, play/pause, and progress (1:00:15 to -37:43). The main content area displays a list of bullet points under the heading "Single Expression":
- Lambda functions are limited to a single expression. This means you can't include multiple statements or use `for` loops within a lambda function.
  - The expression's result is automatically returned by the lambda function.
- Below this, another section titled "Implicit Return:" is shown with the following bullet points:
- Lambda functions automatically return the result of evaluating the expression.
  - There's no need to use the `return` keyword explicitly.
- At the bottom of the screen, there is a footer bar with the 'GO CLASSES' logo and the website address [www.goclasses.in](http://www.goclasses.in).

■ What does this print?

```
def do_twice(n, fn): → higher order functions
    return fn(fn(n))

print(do_twice(3, lambda x: x**2))
```

*a function which takes function as an argument*

$3^2 = 9$        $9^2 = \underline{\underline{81}}$

*✓*

[https://ocw.mit.edu/courses/6-100L-introduction-to-CS-and-programming-using-Python-fall-2022/mit6\\_100L\\_f22\\_lec09.pdf](https://ocw.mit.edu/courses/6-100L-introduction-to-CS-and-programming-using-Python-fall-2022/mit6_100L_f22_lec09.pdf)

Host  
1:07:10 - 30:48

```
def do_twice(n, fn): 9
    return fn(fn(n))

print(do_twice(3, lambda x: x**2))
```

Global environment

do_twice	function object
----------	-----------------

do\_twice environment

n	3
fn	lambda x: x**2

lambda x: x\*\*2 environment

x	9
---	---

lambda x: x\*\*2 environment

x	3
---	---

Returns 9

10  
6.100L Lecture 9

www.goclasses.in

The screenshot shows a Go Classes video player interface. In the top right corner, there is a video feed of a host. Below the video, the word "Host" is written. The main area displays a code execution diagram. On the left, under "Global environment", there is a function definition: `def do_twice(n, fn): return fn(fn(n))`. A blue box highlights the call `print(do_twice(3, lambda x: x**2))`. To the right, under "do\_twice environment", the variables are shown: `n = 3` and `fn = lambda x: x**2`. A blue box highlights the output "Returns 81". A yellow arrow points from the "PRINTS 81" text to the "Returns 81" text. At the bottom of the diagram, the number "12" is visible. Below the diagram, a video player bar shows the URL "www.goclasses.in", the current time "1:07:20", and the total duration "-30:38".

The screenshot shows a Go Classes video player interface. In the top right corner, there is a video feed of a host. Below the video, the word "Host" is written. The main area displays a code execution diagram. On the left, there is a code snippet: `def inc_maker(i): return lambda x:x+i`. Below it, a line of code is shown: `>>> inc_maker(3)(4)`. A yellow oval surrounds the part `inc_maker(3)`, and a yellow arrow points down to the text "it is a function". To the right, a Jupyter Notebook cell is shown with the input "In [38]: def inc\_maker(i): return lambda x:x+i inc\_maker(3)(4)" and the output "Out[38]: 7". At the bottom of the diagram, the number "12" is visible. Below the diagram, a video player bar shows the URL "www.goclasses.in".

