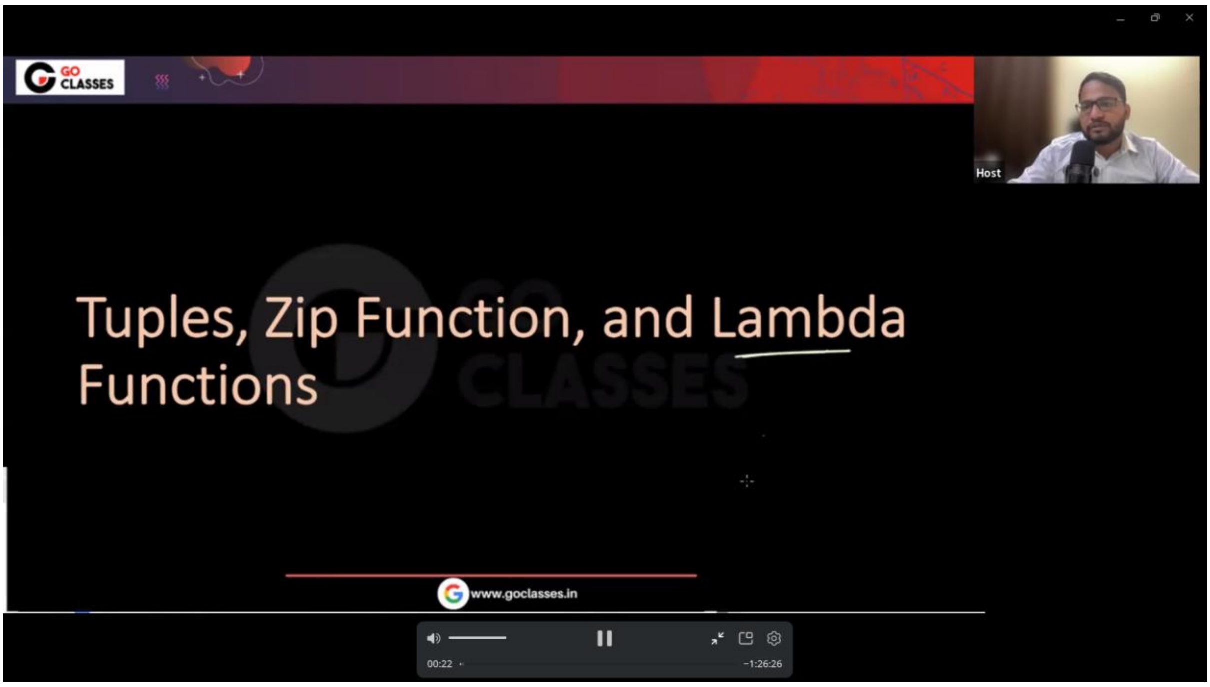
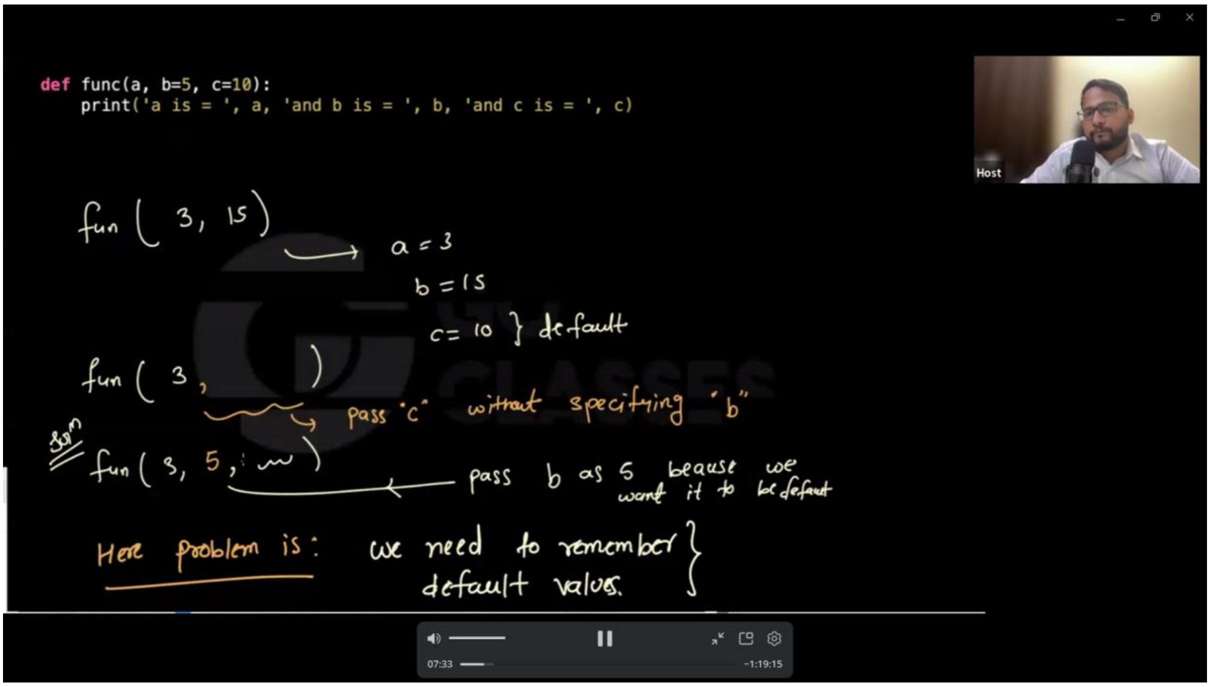


Tuples, Zip Function, and Lambda Functions



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

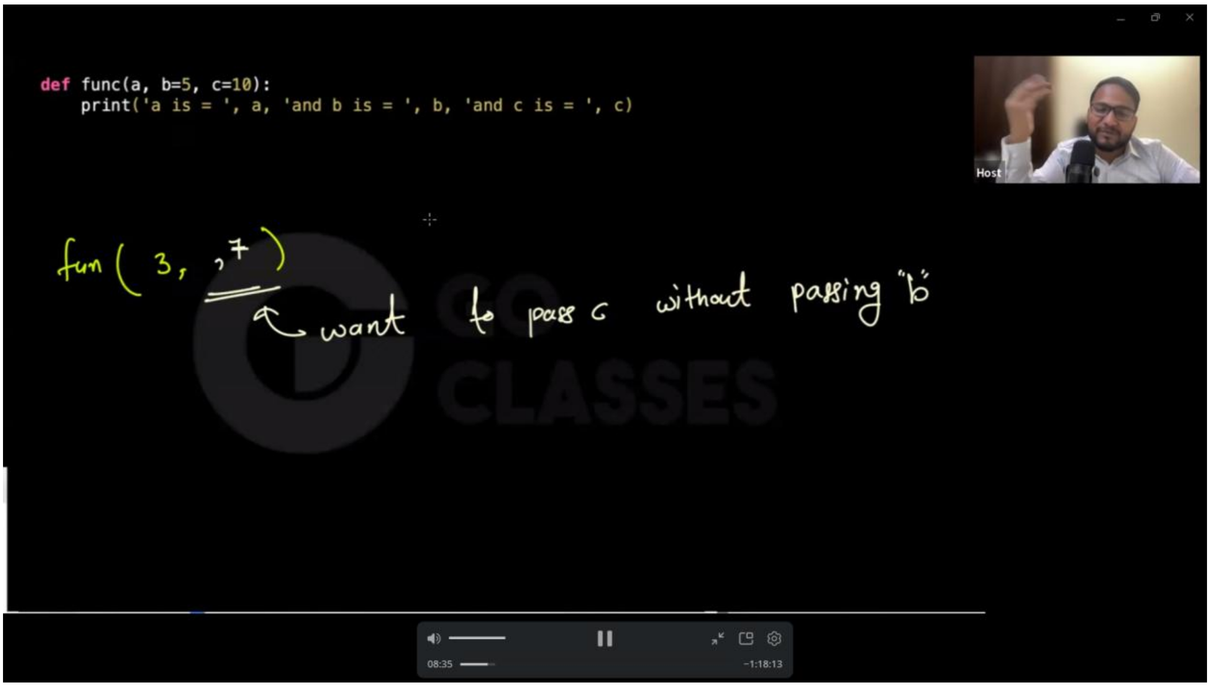
fun (3, 15) → a = 3
b = 15
c = 10 } default

fun (3,) → pass "c" without specifying "b"

Soln

fun (3, 5, ~) ← pass b as 5 beause we want it to be default

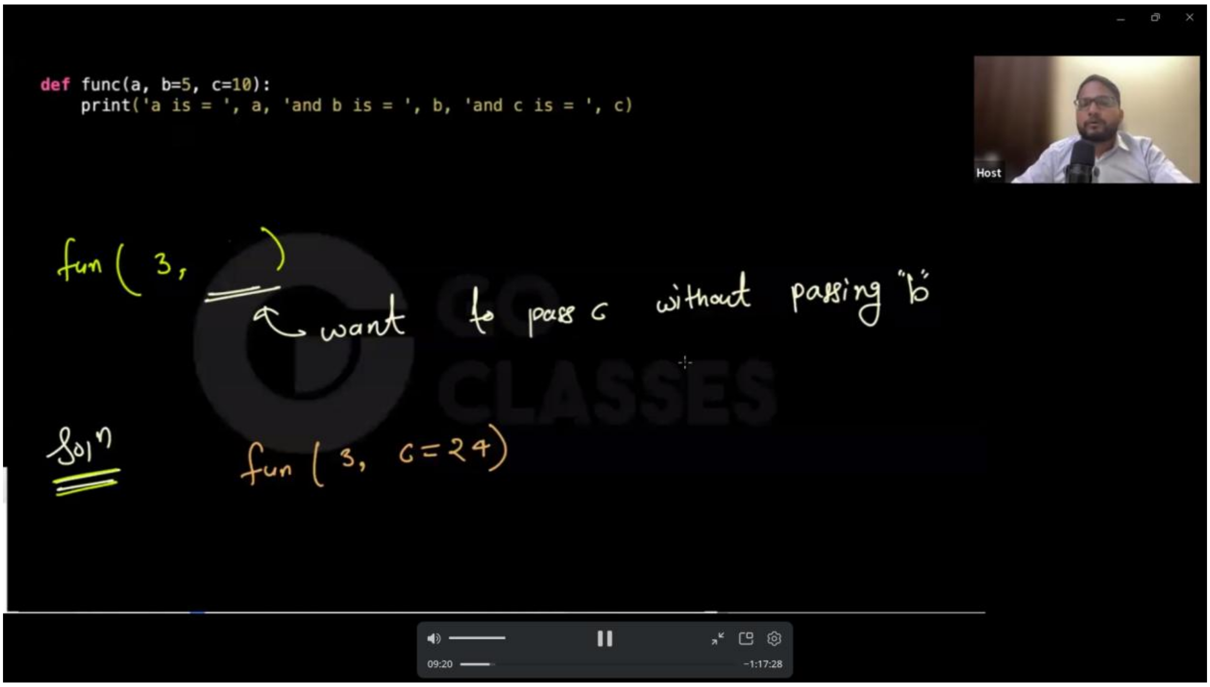
Here problem is : we need to remember default values. }



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

fun (3, ,7)

want to pass c without passing "b"



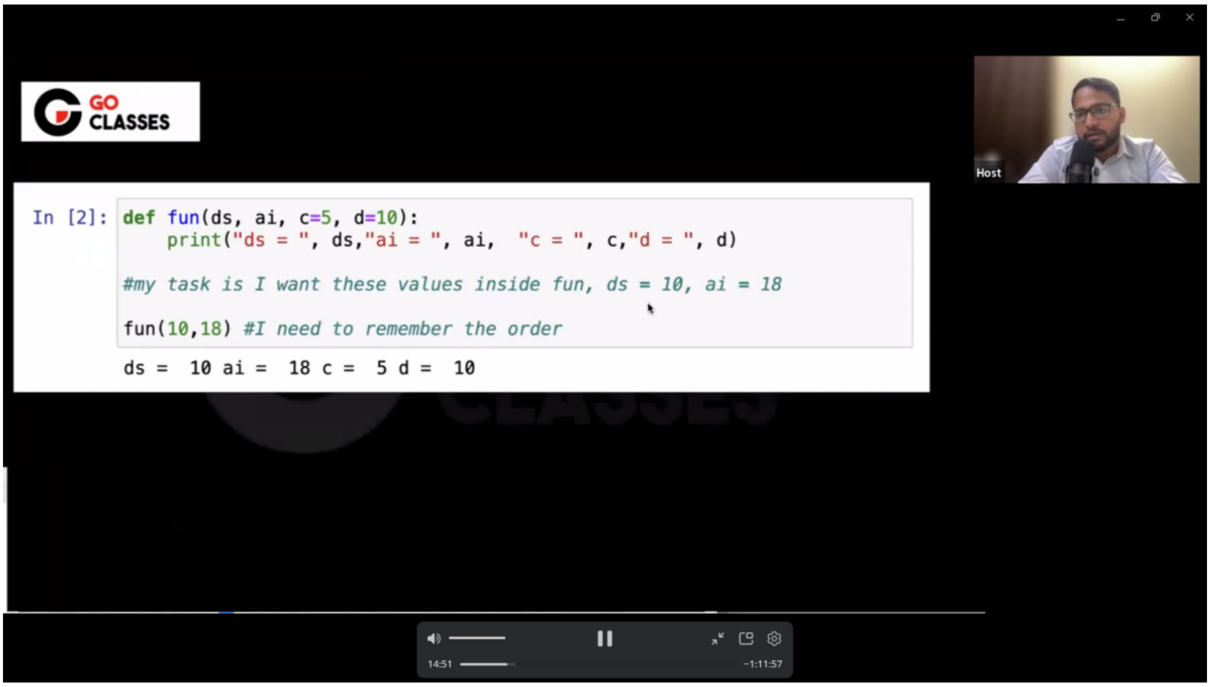
```
def func(a, b=5, c=10):
    print('a is = ', a, 'and b is = ', b, 'and c is = ', c)
```

fun (3,)

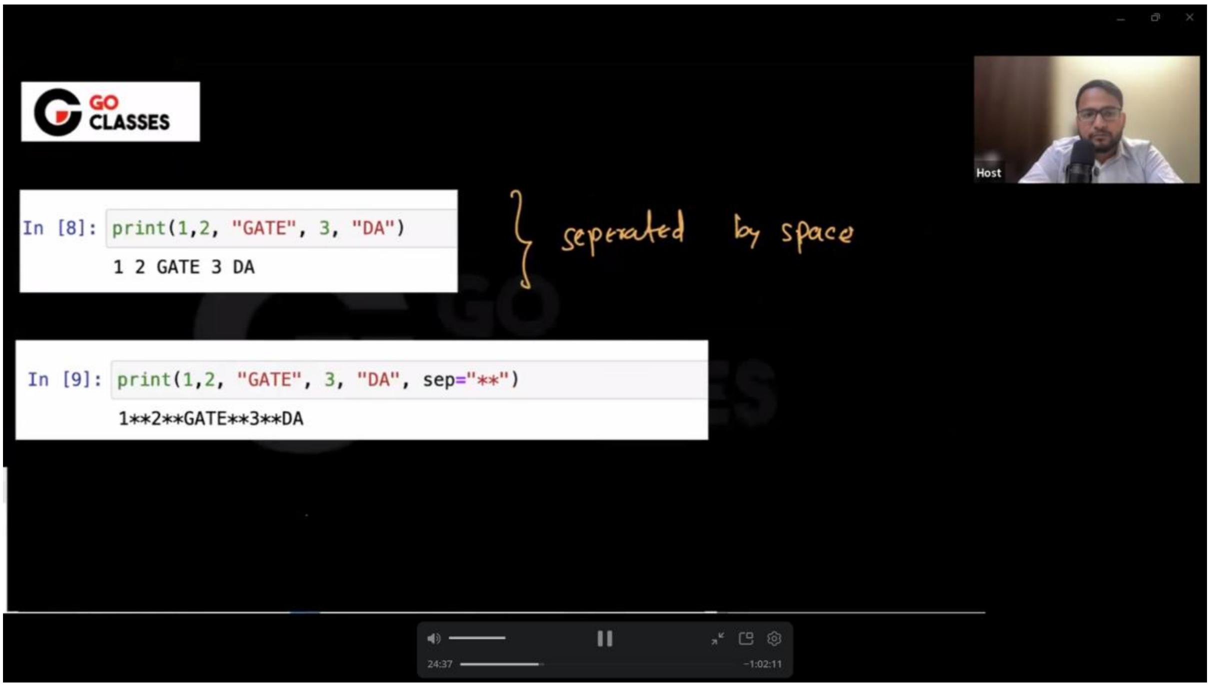
want to pass c without passing "b"

sol^n

fun (3, c=24)



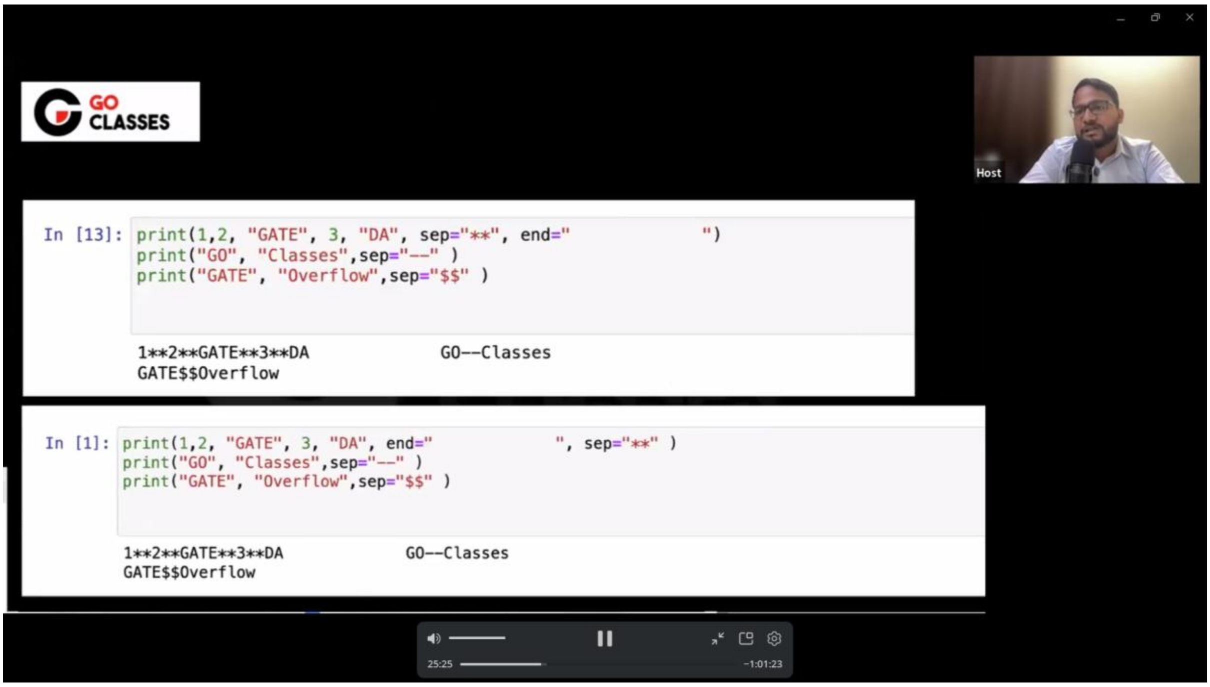
```
In [2]: def fun(ds, ai, c=5, d=10):
            print("ds = ", ds,"ai = ", ai,  "c = ", c,"d = ", d)

        #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
```



```
In [8]: print(1,2, "GATE", 3, "DA")

        1 2 GATE 3 DA
```

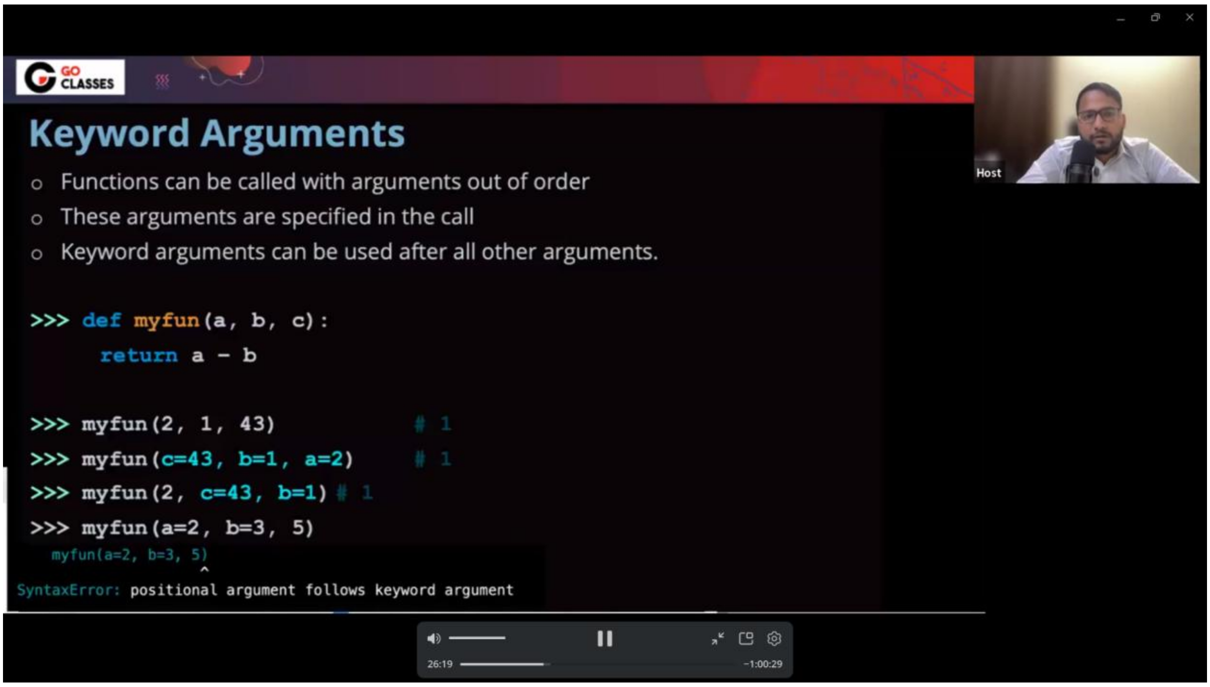
} seperated by space

```
In [9]: print(1,2, "GATE", 3, "DA", sep="**")

        1**2**GATE**3**DA
```



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

         1**2**GATE**3**DA               GO--Classes
         GATE$$Overflow
```

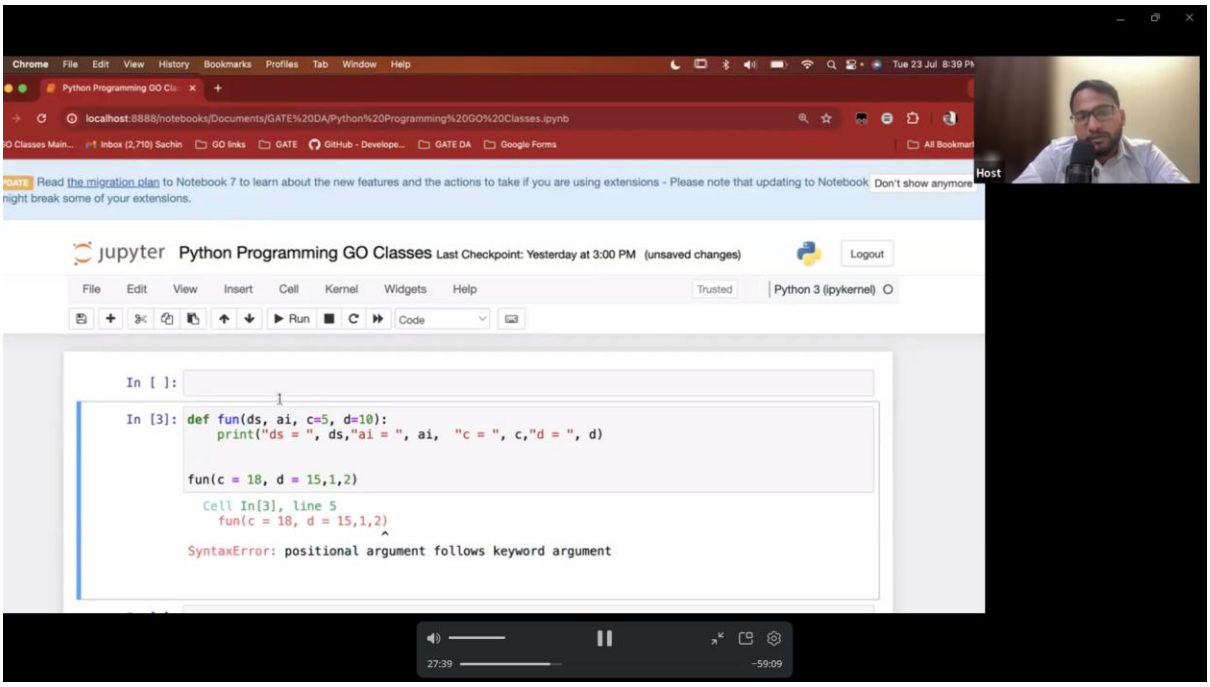
```
In [1]: print(1,2, "GATE", 3, "DA", end="               ", sep="**" )
        print("GO", "Classes",sep="--" )
        print("GATE", "Overflow",sep="$$" )

        1**2**GATE**3**DA               GO--Classes
        GATE$$Overflow
```



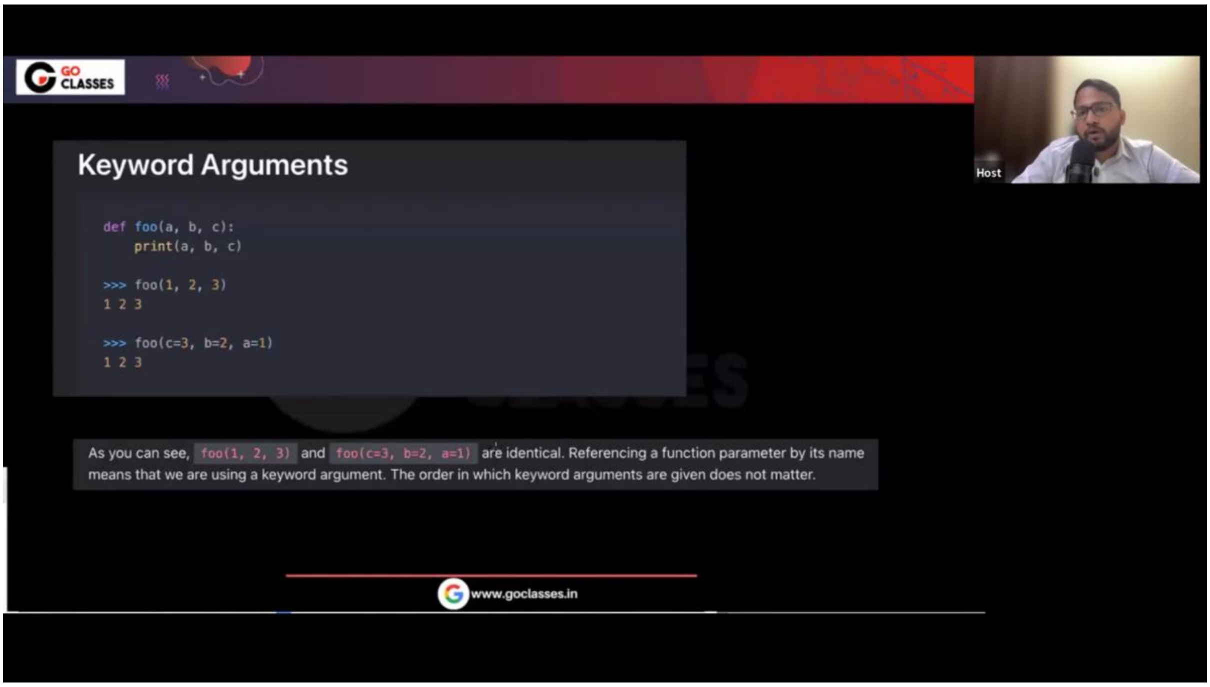
Keyword Arguments

- Functions can be called with arguments out of order
- These arguments are specified in the call
- 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
```


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue 23 Jul 8:39 PM
Python Programming GO Cla
localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb
Inbox (2,710) Sachin
GO links
GATE
GitHub - Develope...
GATE DA
Google Forms
All Bookmarks
Host
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
might break some of your extensions.
Don't show anymore
jupyter
Python Programming GO Classes
Last Checkpoint: Yesterday at 3:00 PM (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3 (ipykernel)
Run
Code
27:39
-59:09

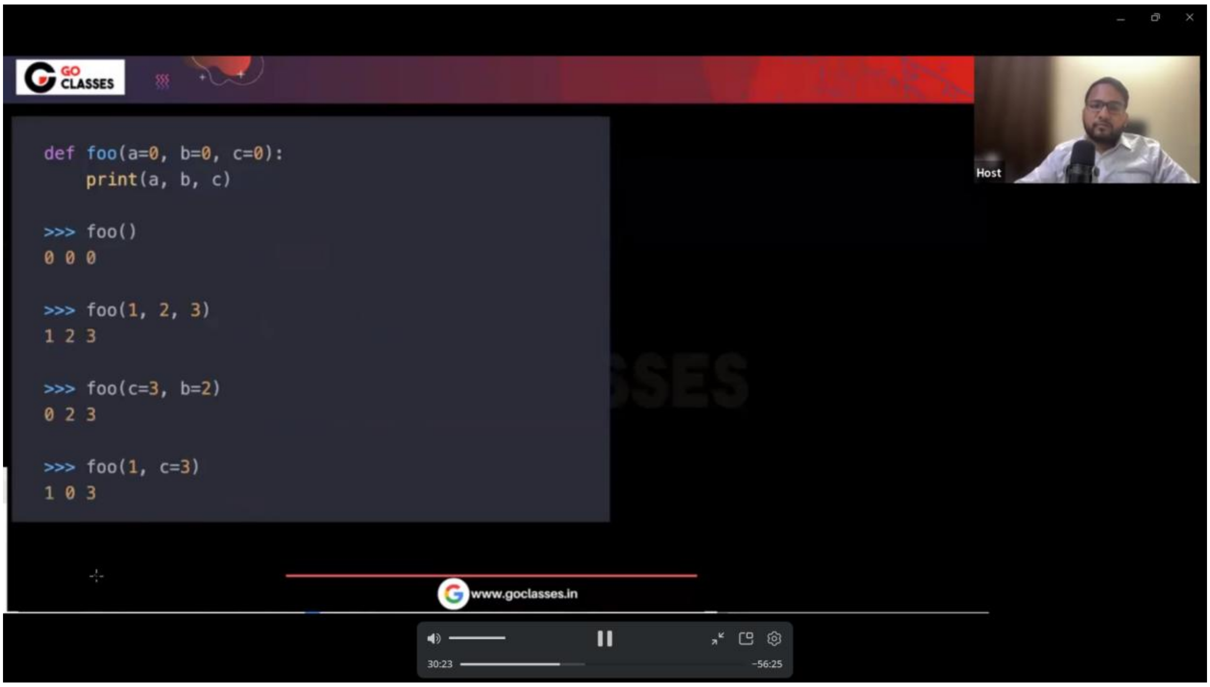
```
In [ ]:

In [3]: def fun(ds, ai, c=5, d=10):
            print("ds = ", ds,"ai = ", ai,  "c = ", c,"d = ", d)


        fun(c = 18, d = 15,1,2)

          Cell In[3], line 5
            fun(c = 18, d = 15,1,2)
                                 ^
        SyntaxError: positional argument follows keyword argument
```



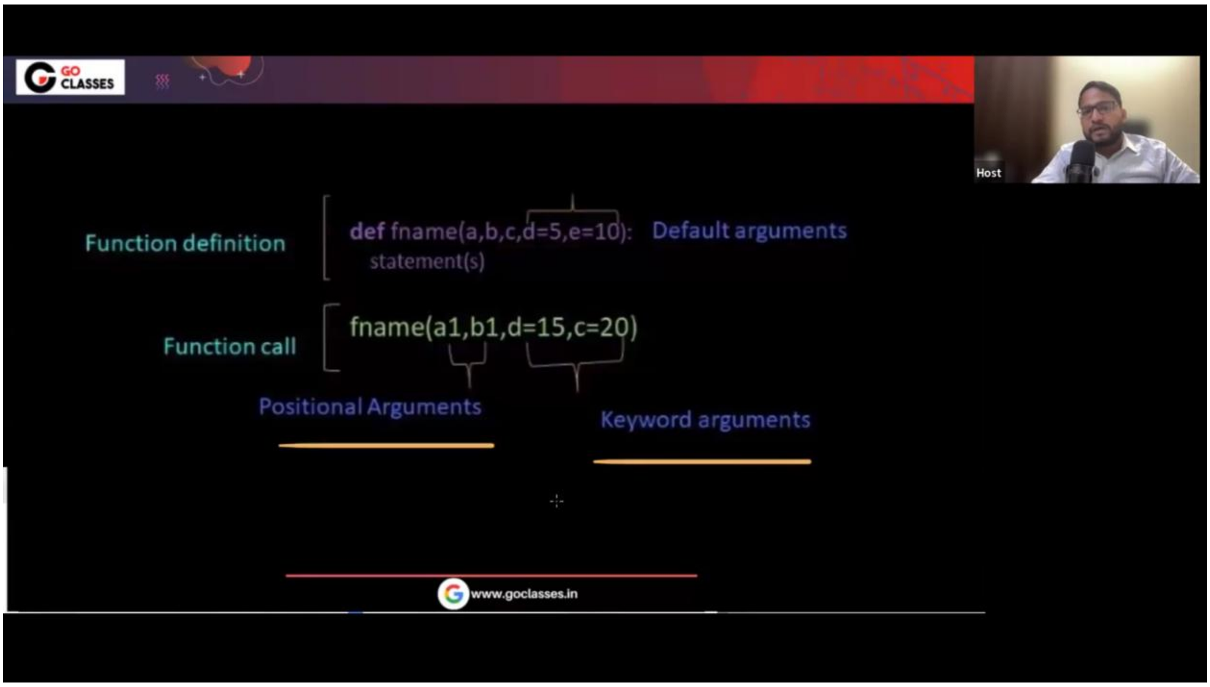
Keyword Arguments

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

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

>>> foo(c=3, b=2, a=1)
1 2 3
```

As you can see, `foo(1, 2, 3)` and `foo(c=3, b=2, a=1)` are identical. Referencing a function parameter by its name means that we are using a keyword argument. The order in which keyword arguments are given does not matter.



```
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
```



Function definition

```
def fname(a,b,c,d=5,e=10):
  statement(s)
```

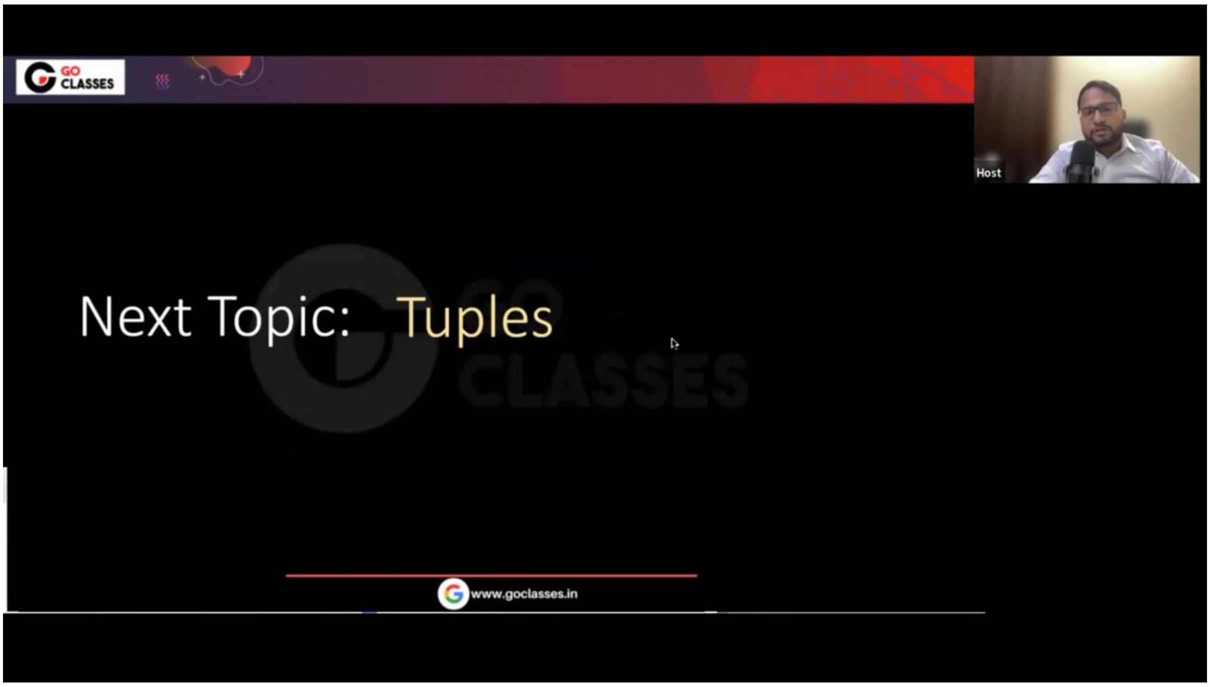
Default arguments

Function call

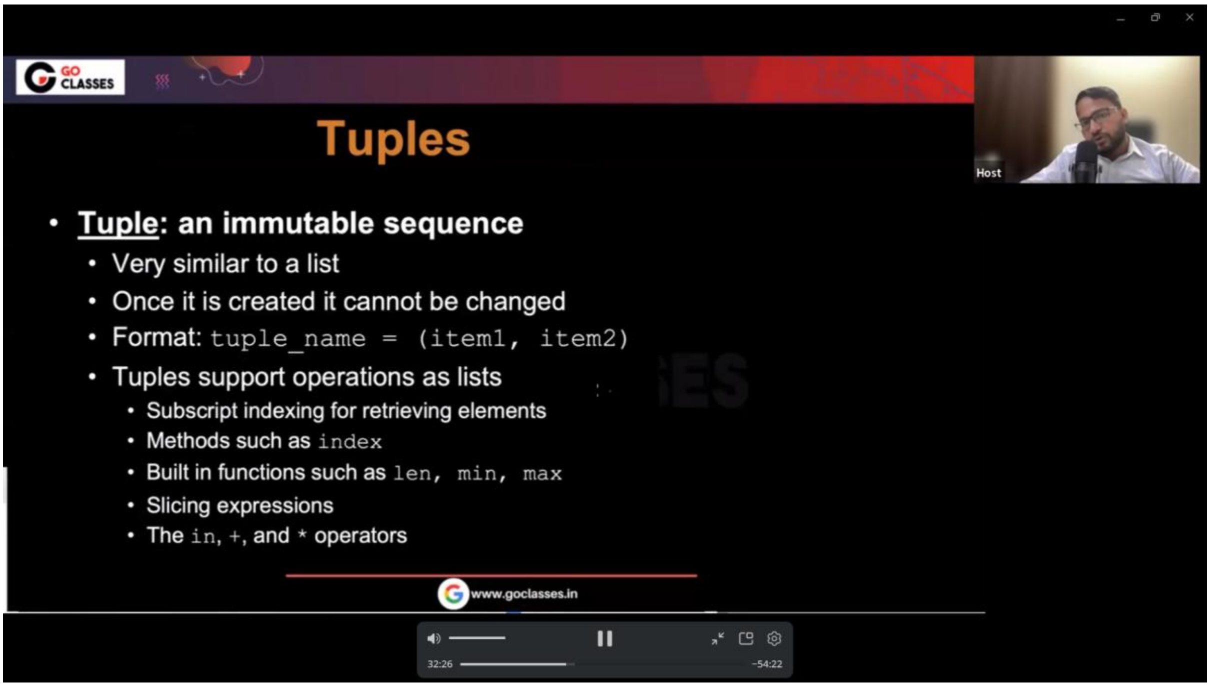
```
fname(a1,b1,d=15,c=20)
```

Positional Arguments

Keyword arguments

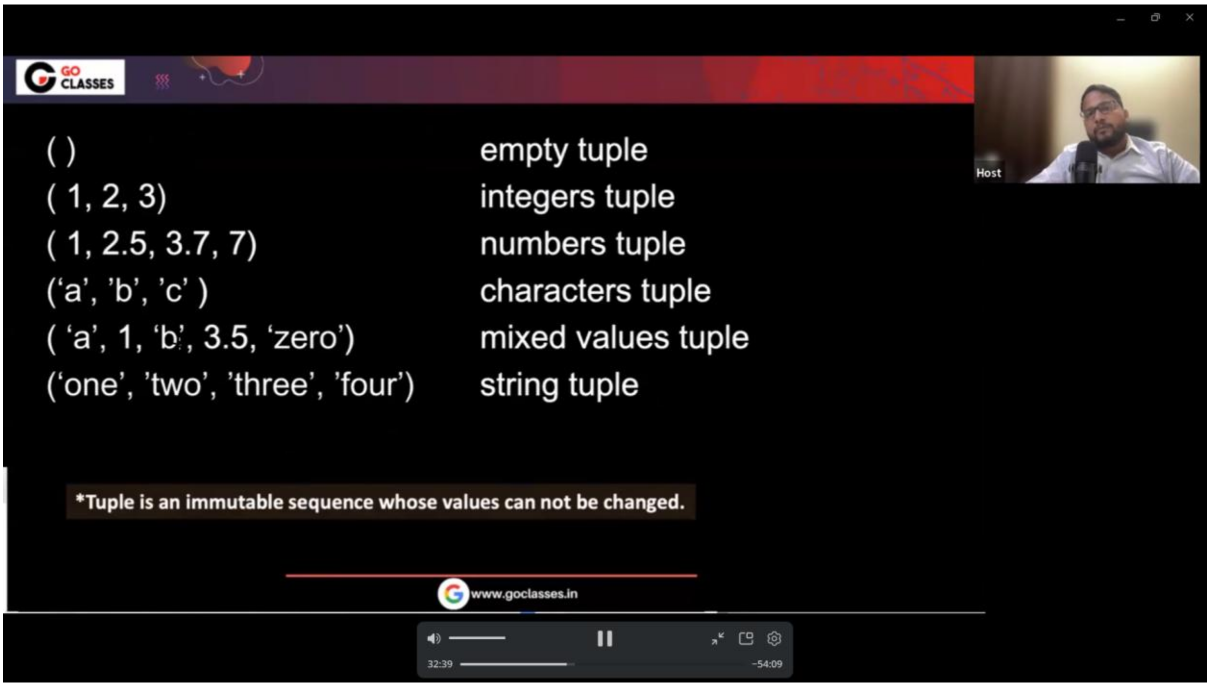


Next Topic: Tuples



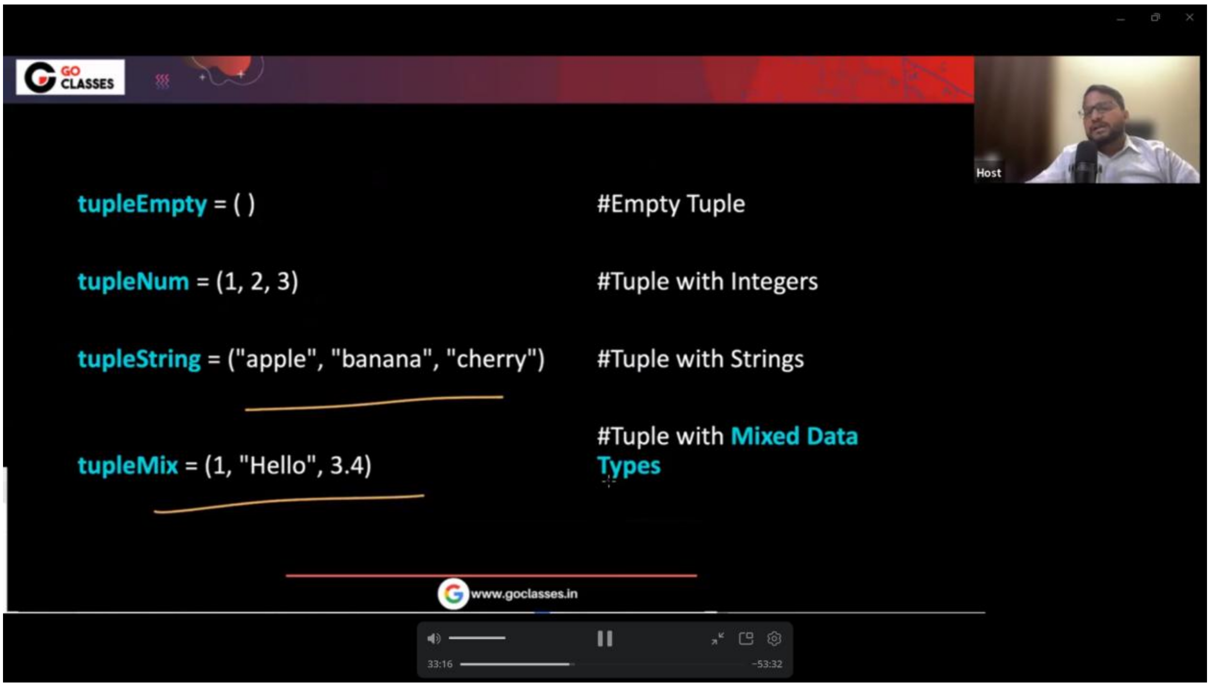
Tuples

- **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

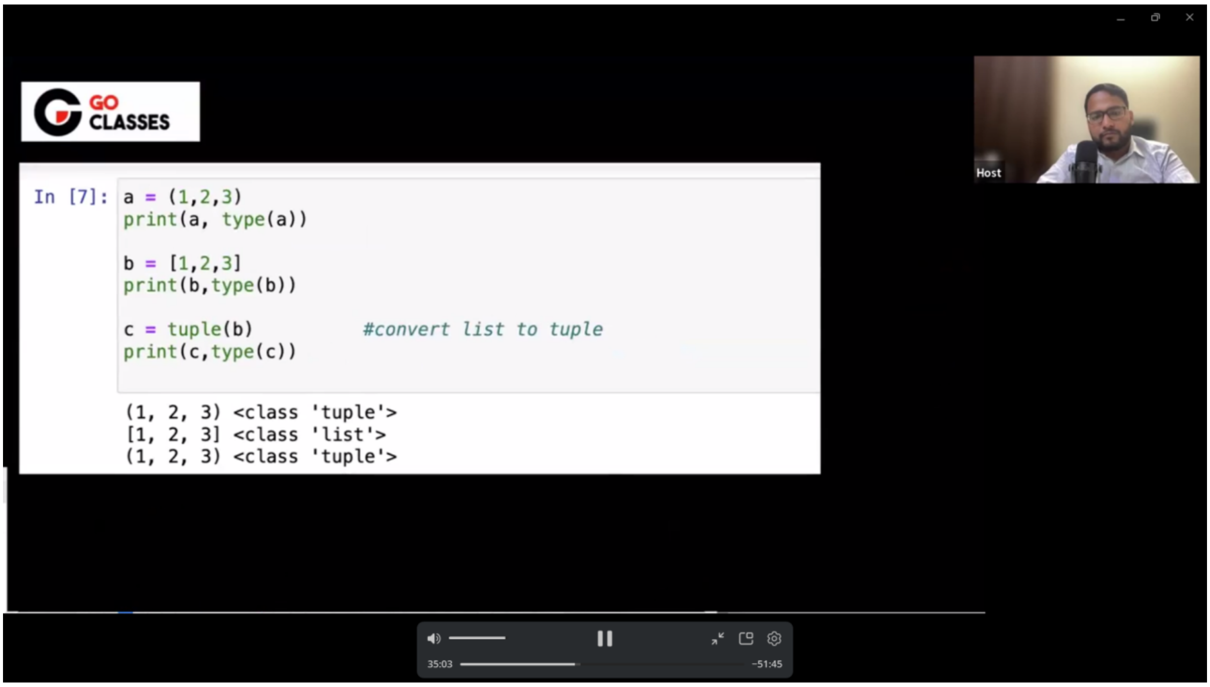


( )	empty tuple
(1, 2, 3)	integers tuple
(1, 2.5, 3.7, 7)	numbers tuple
(‘a’, ’b’, ’c’)	characters tuple
(‘a’, 1, ‘b’, 3.5, ‘zero’)	mixed values tuple
(‘one’, ’two’, ’three’, ’four’)	string tuple

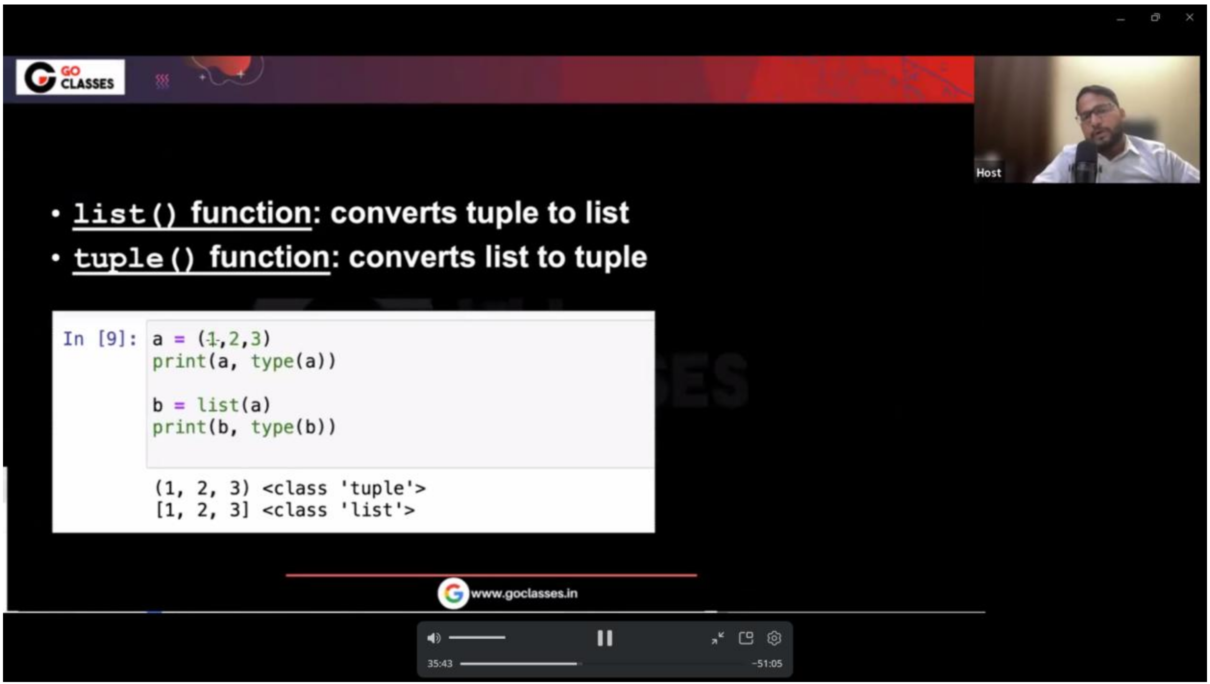
*Tuple is an immutable sequence whose values can not be changed.



```
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
```

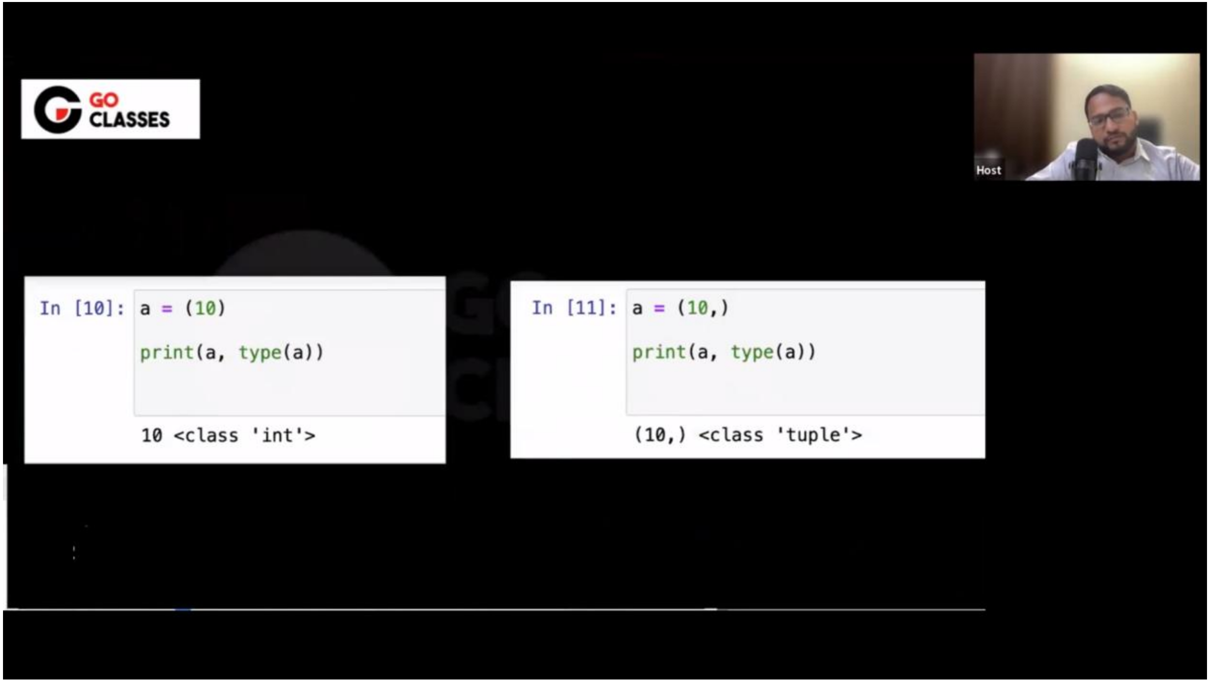


```
In [7]: a = (1,2,3)
        print(a, type(a))

        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'>
```



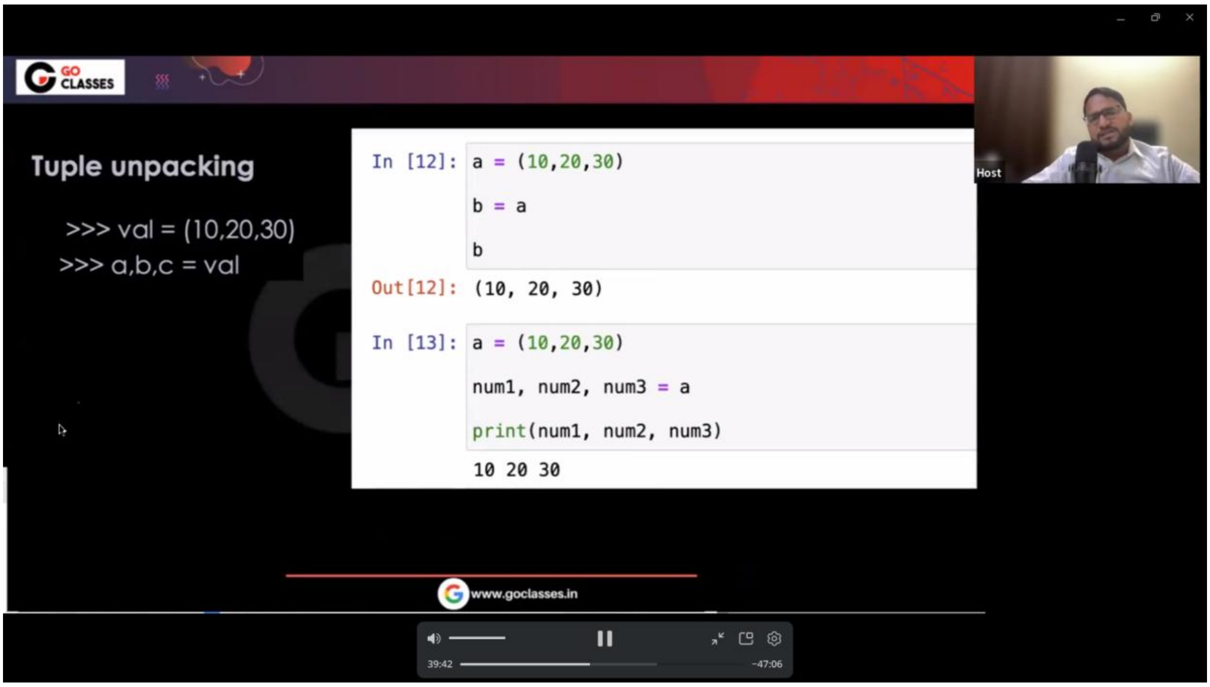
- `list()` **function**: converts tuple to list
- `tuple()` **function**: converts list to tuple

```
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'>
```

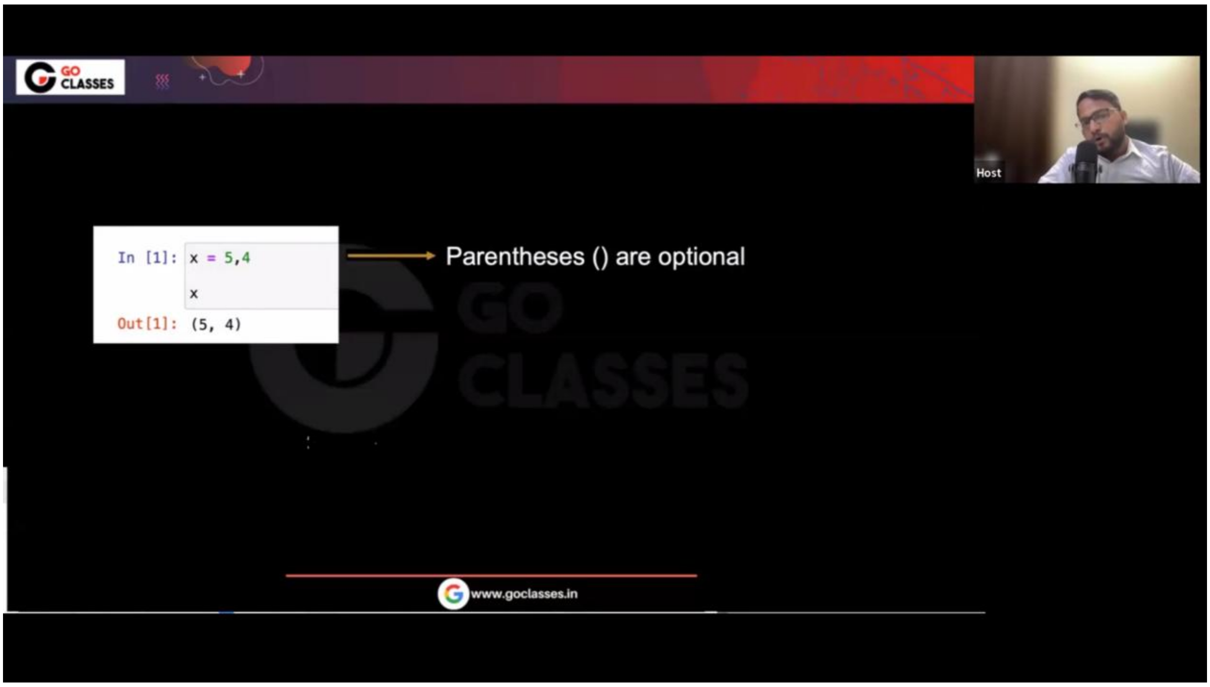
```
In [11]: a = (10,)

         print(a, type(a))

         (10,) <class 'tuple'>
```



Tuple unpacking

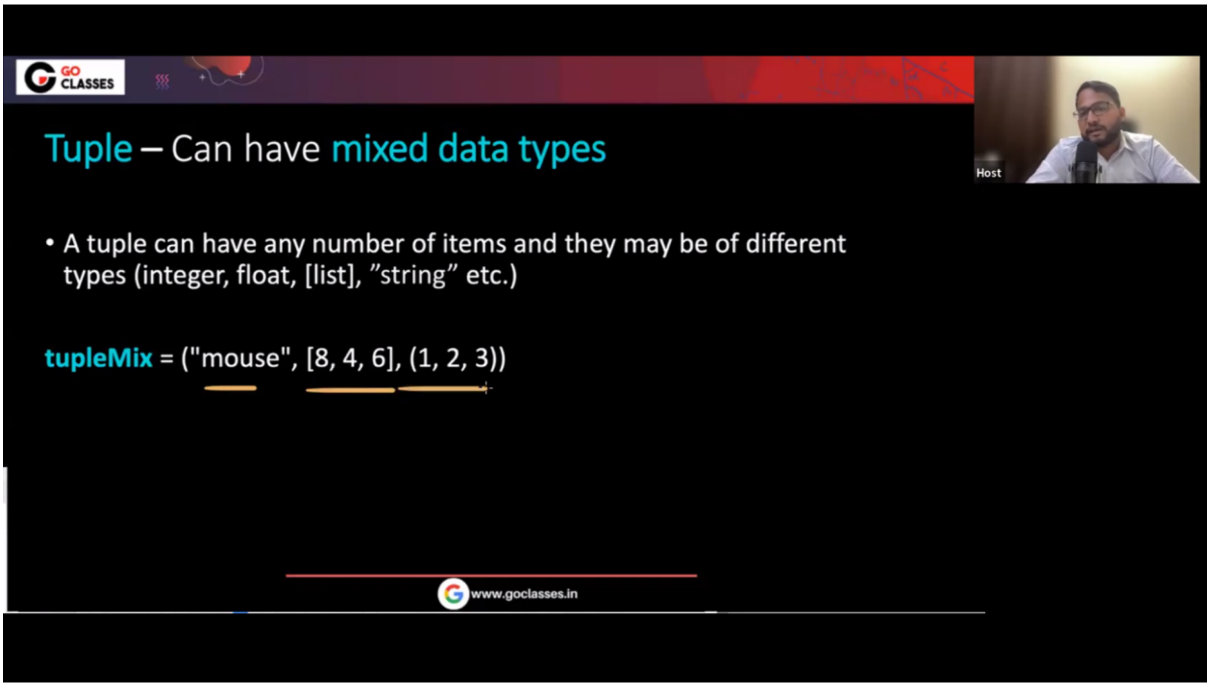
```
>>> val = (10,20,30)
>>> a,b,c = val
```

```
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
```



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

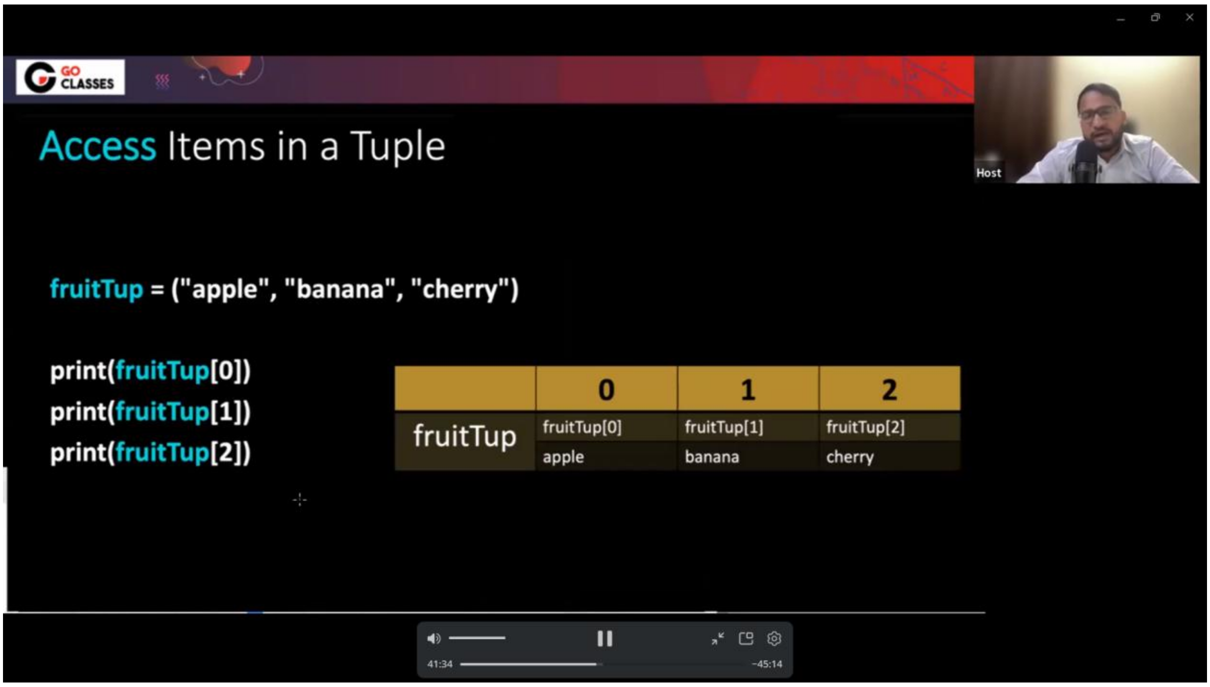
Parentheses () are optional



Tuple – Can have mixed data types

- A tuple can have any number of items and they may be of different types (integer, float, [list], "string" etc.)

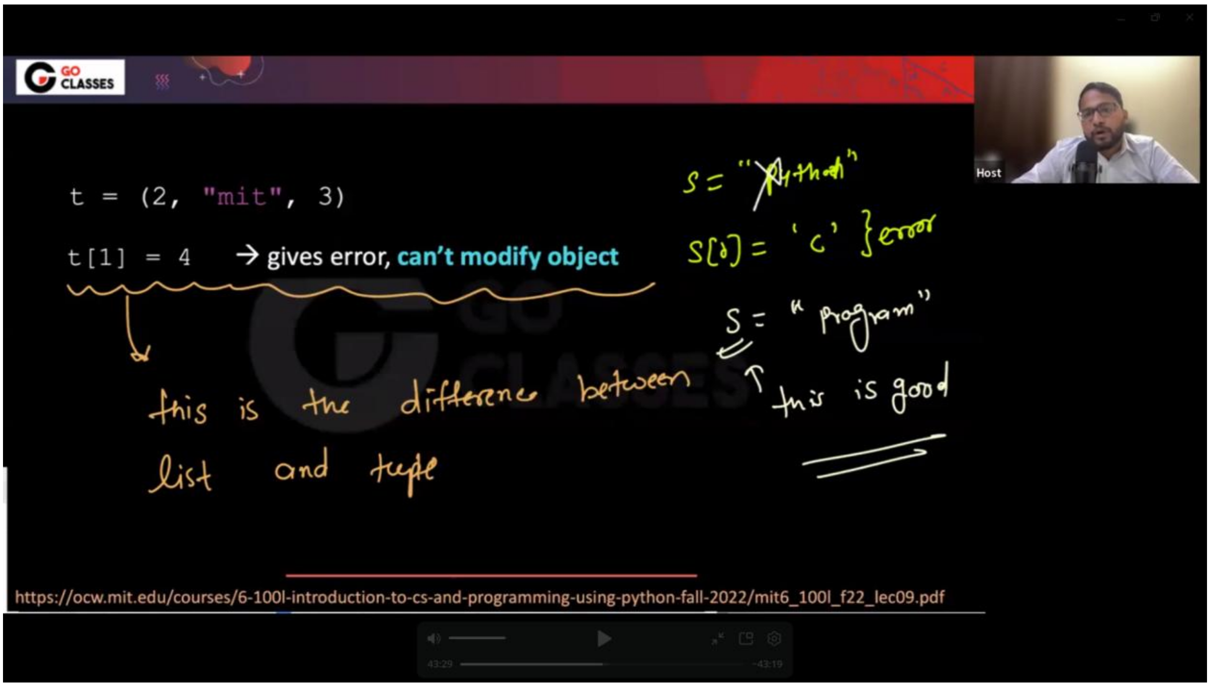
```
tupleMix = ("mouse", [8, 4, 6], (1, 2, 3))
```



Access Items in a Tuple

```
fruitTup = ("apple", "banana", "cherry")

print(fruitTup[0])
print(fruitTup[1])
print(fruitTup[2])
```

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



```
t = (2, "mit", 3)

t[1] = 4
```

→ gives error, **can't modify object**

this is the difference between list and tuple

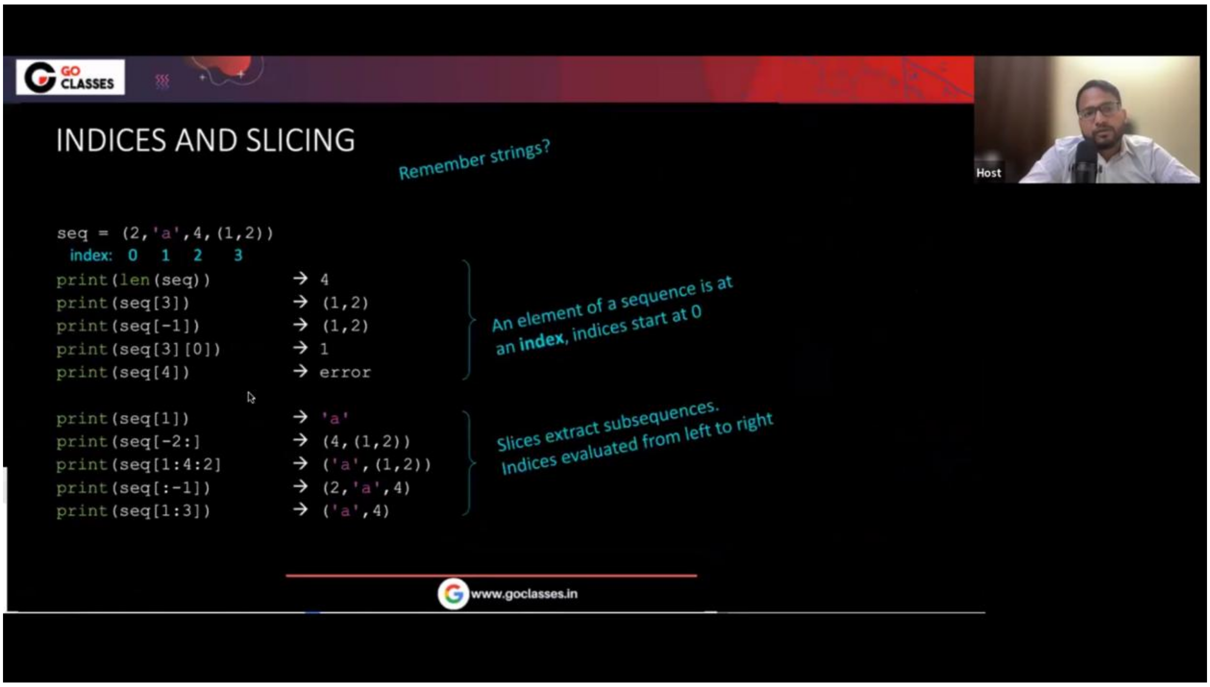
S = "~~P~~ython"

S[0] = 'c' } error

S = "program"

this is good

https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6_100l_f22_lec09.pdf



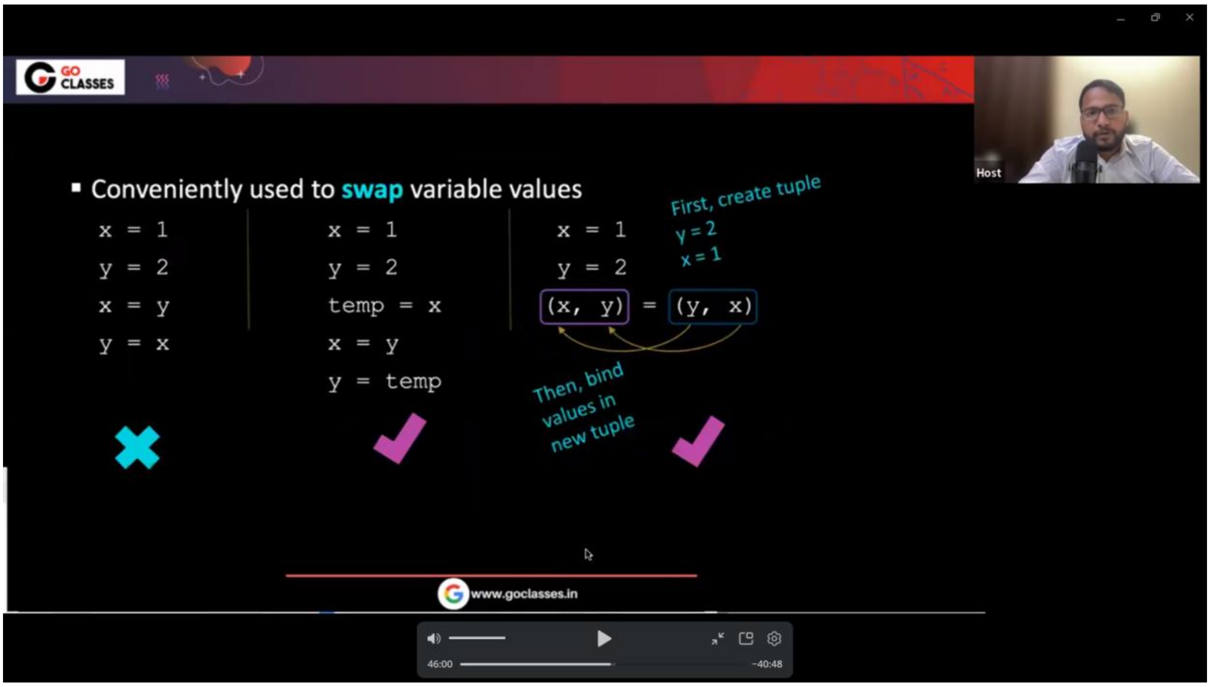
INDICES AND SLICING

Remember strings?

```
seq = (2,'a',4,(1,2))
  index:  0   1   2    3
print(len(seq))          → 4
print(seq[3])            → (1,2)
print(seq[-1])           → (1,2)
print(seq[3][0])         → 1
print(seq[4])            → error

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)
```

An element of a sequence is at an **index**, indices start at 0

Slices extract subsequences. Indices evaluated from left to right



- Conveniently used to **swap** variable values

```
x = 1
y = 2
x = y
y = x
```

✗

```
x = 1
y = 2
temp = x
x = y
y = temp
```

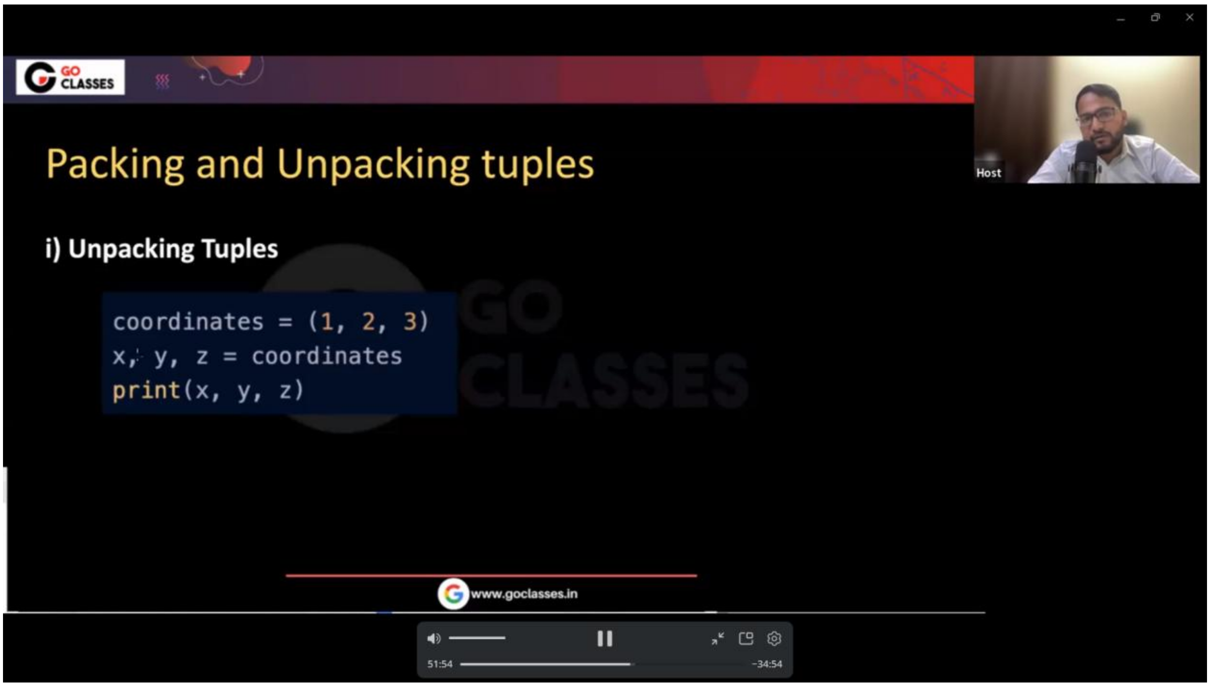
✓

```
x = 1
y = 2
(x, y) = (y, x)
```

First, create tuple
y = 2
x = 1

Then, bind values in new tuple

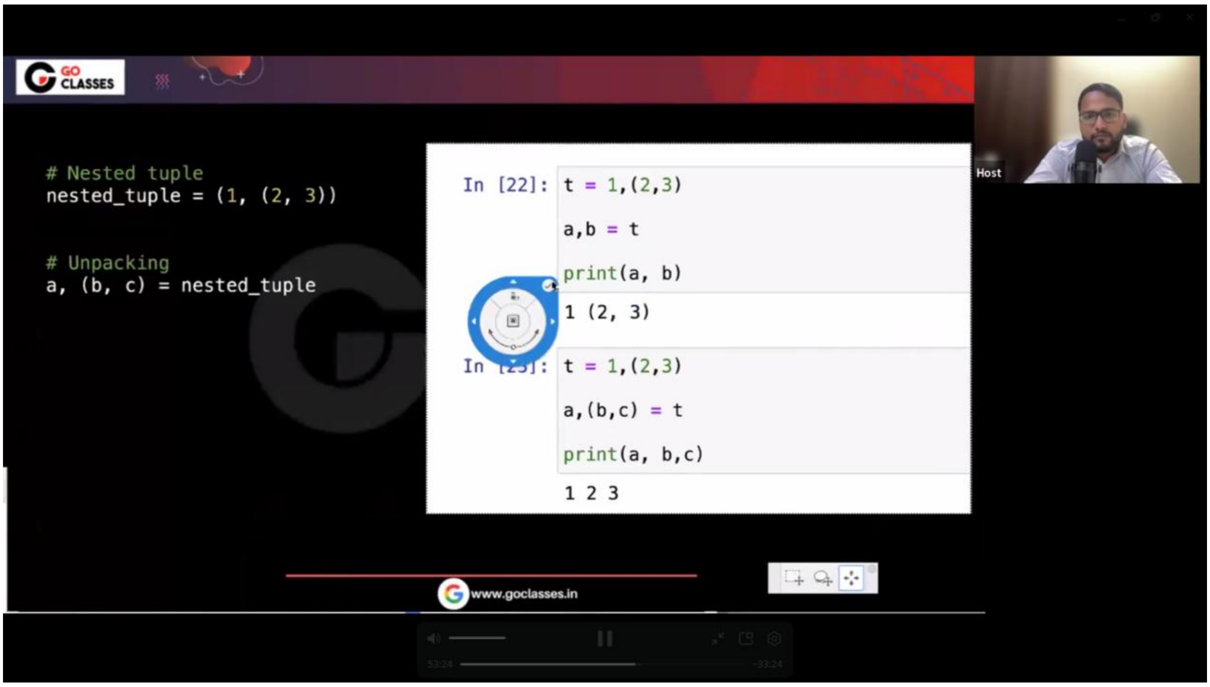
✓



Packing and Unpacking tuples

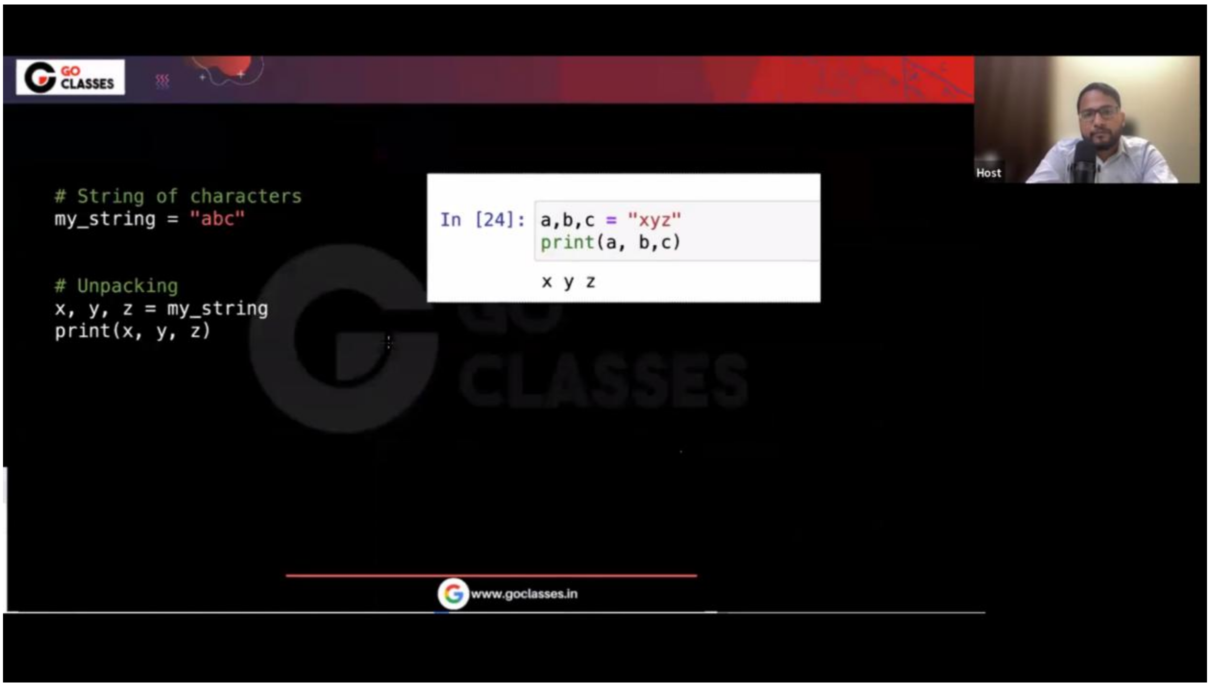
i) Unpacking Tuples

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



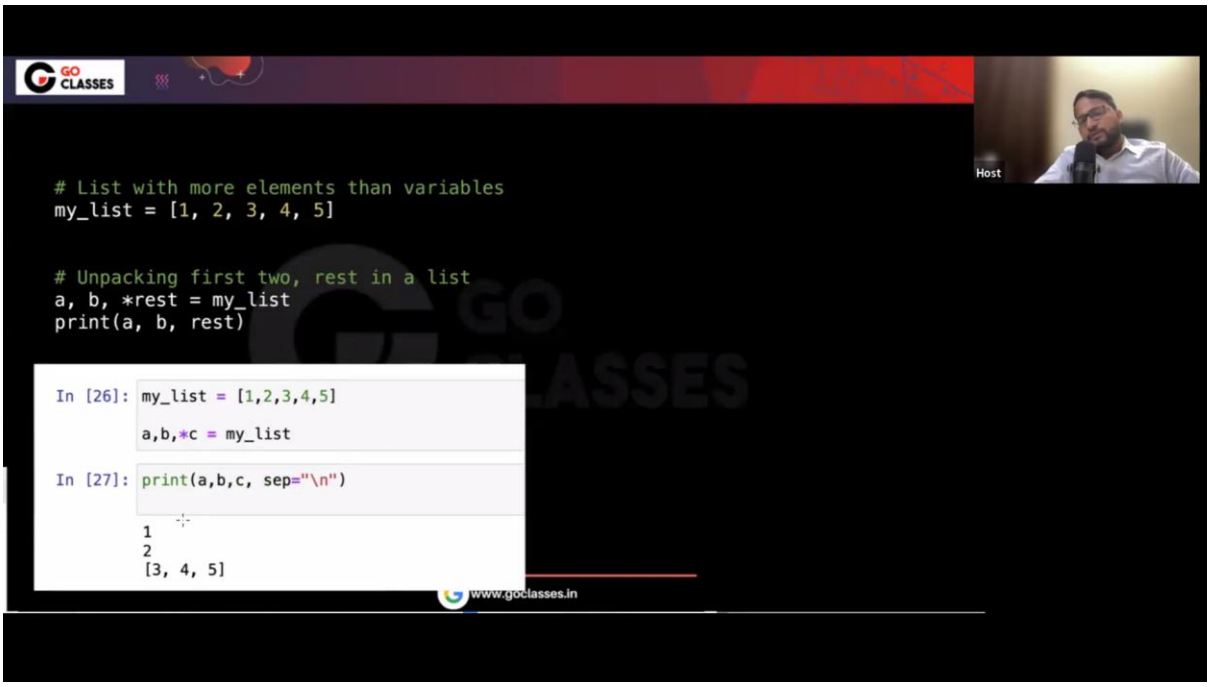
```
# Nested tuple
nested_tuple = (1, (2, 3))

# 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
```



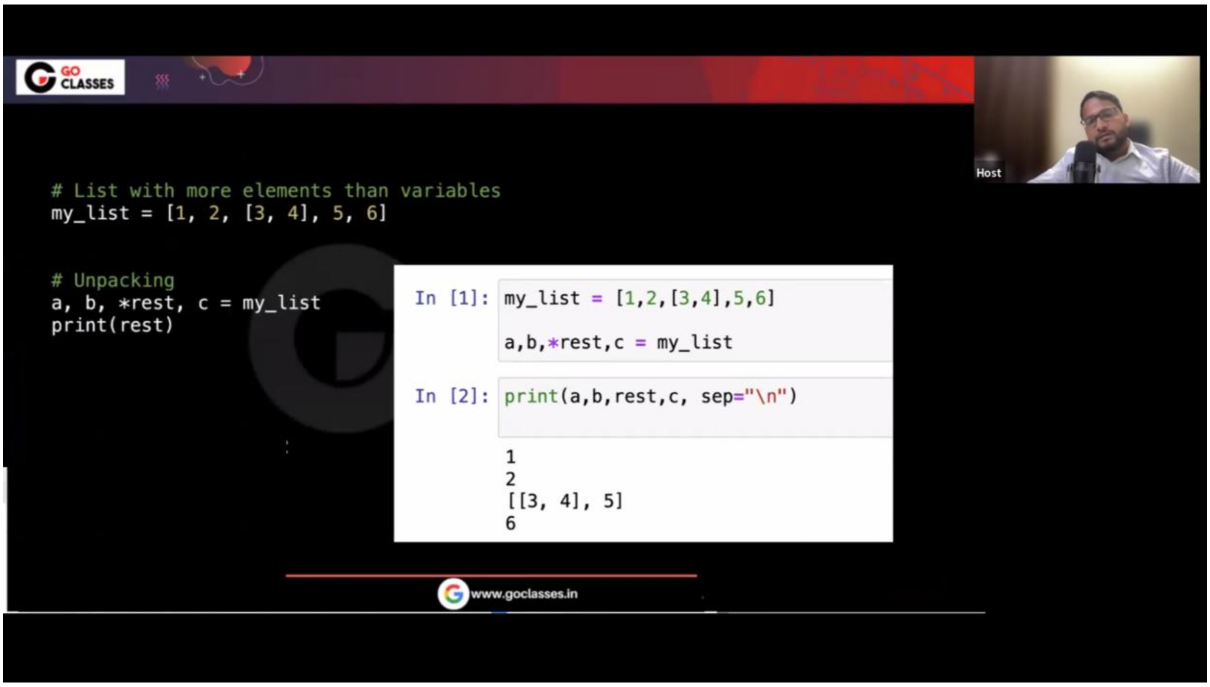
```
# String of characters
my_string = "abc"

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

```
In [24]: a,b,c = "xyz"
         print(a, b,c)

         x y z
```



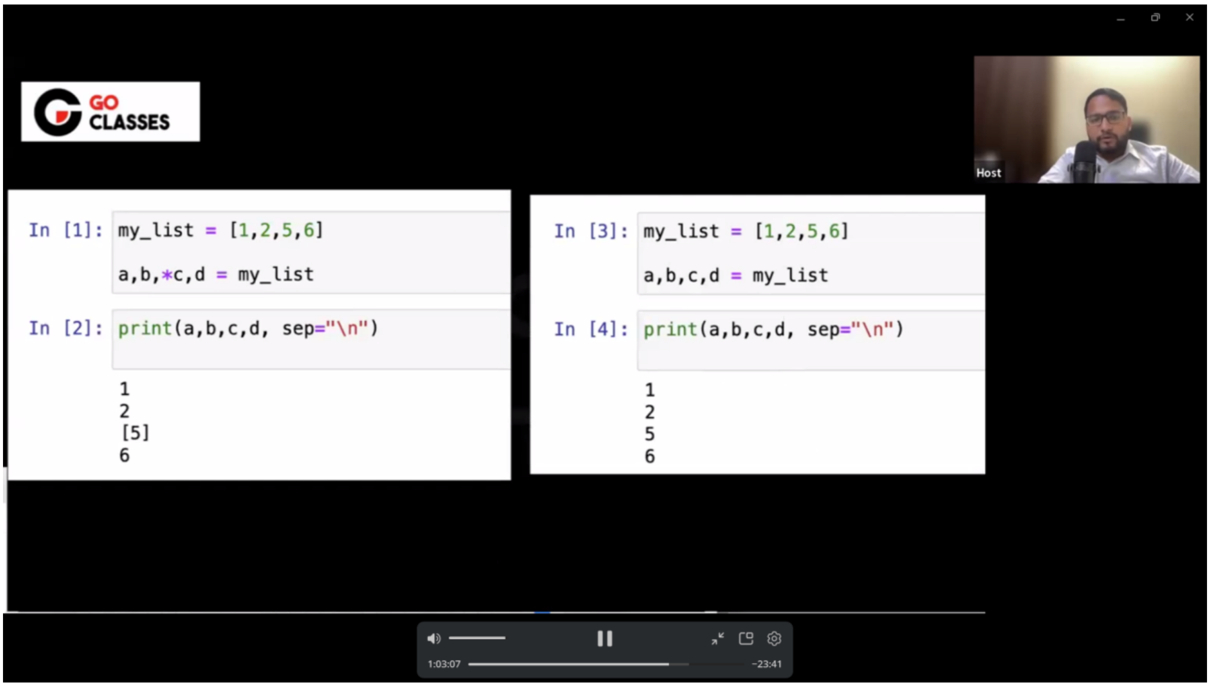
```
# List with more elements than variables
my_list = [1, 2, 3, 4, 5]


# Unpacking first two, rest in a list
a, b, *rest = my_list
print(a, b, rest)
```

```
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]
```



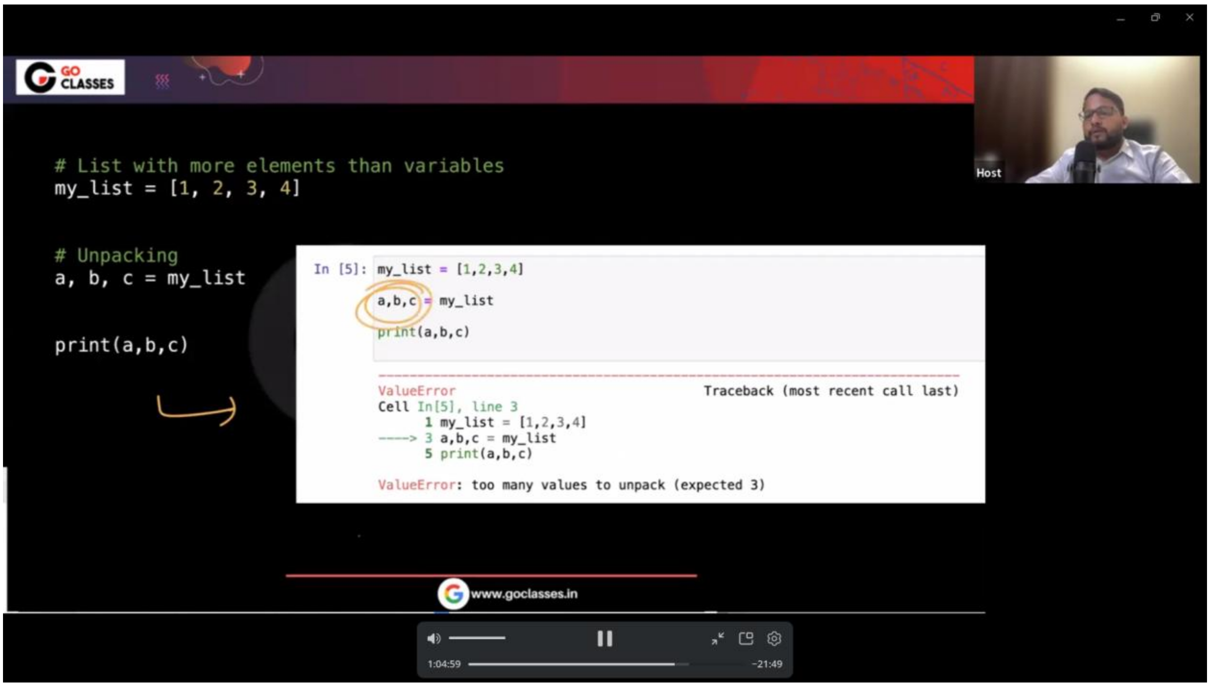
```
# List with more elements than variables
my_list = [1, 2, [3, 4], 5, 6]


# Unpacking
a, b, *rest, c = my_list
print(rest)
```

```
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
```



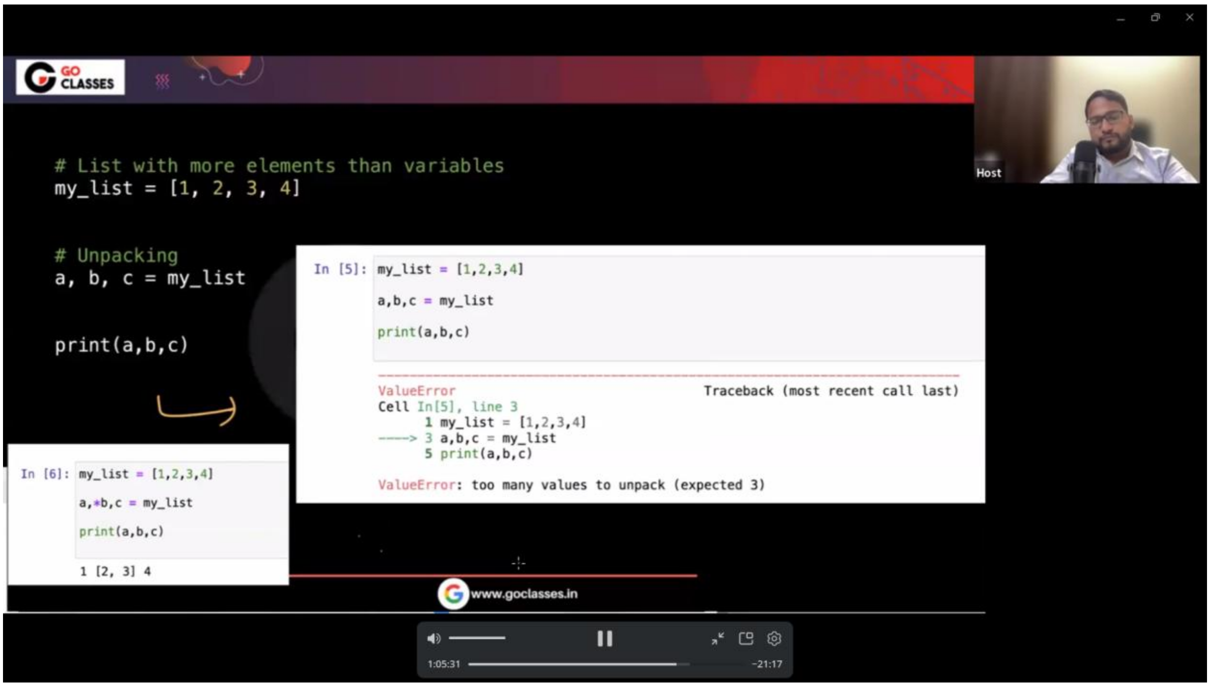
```
In [1]: my_list = [1,2,5,6]

        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
```



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

# Unpacking
a, b, c = my_list

print(a,b,c)
```

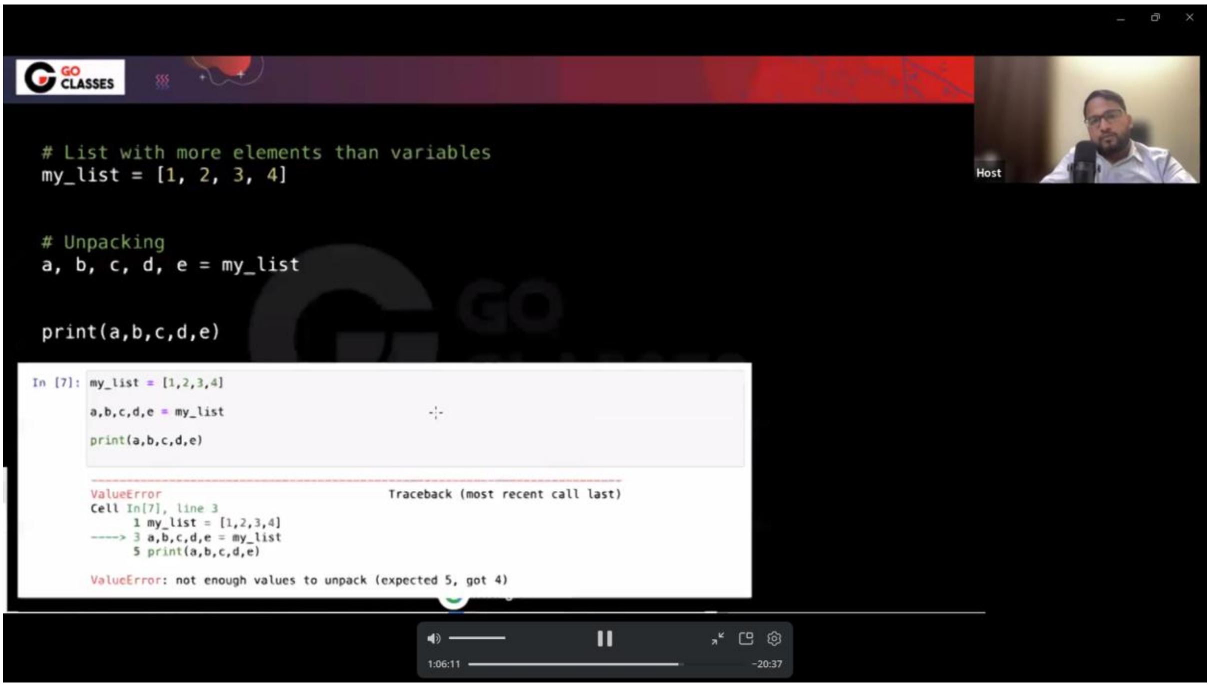
```
In [5]: my_list = [1,2,3,4]

        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]
        ----> 3 a,b,c = my_list
              5 print(a,b,c)

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



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


# Unpacking
a, b, c = my_list


print(a,b,c)
```

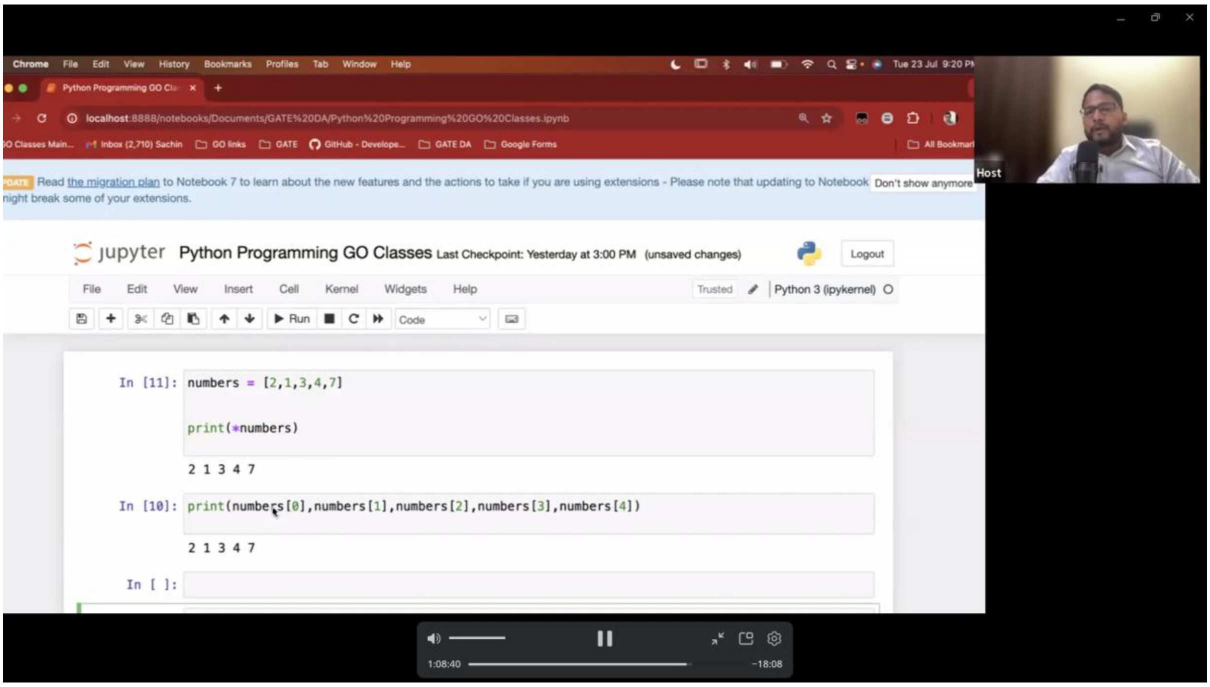
```
In [5]: my_list = [1,2,3,4]

        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]
----> 3 a,b,c = my_list
      5 print(a,b,c)

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

```
In [6]: my_list = [1,2,3,4]

        a,*b,c = my_list

        print(a,b,c)

        1 [2, 3] 4
```



```
# 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)
```

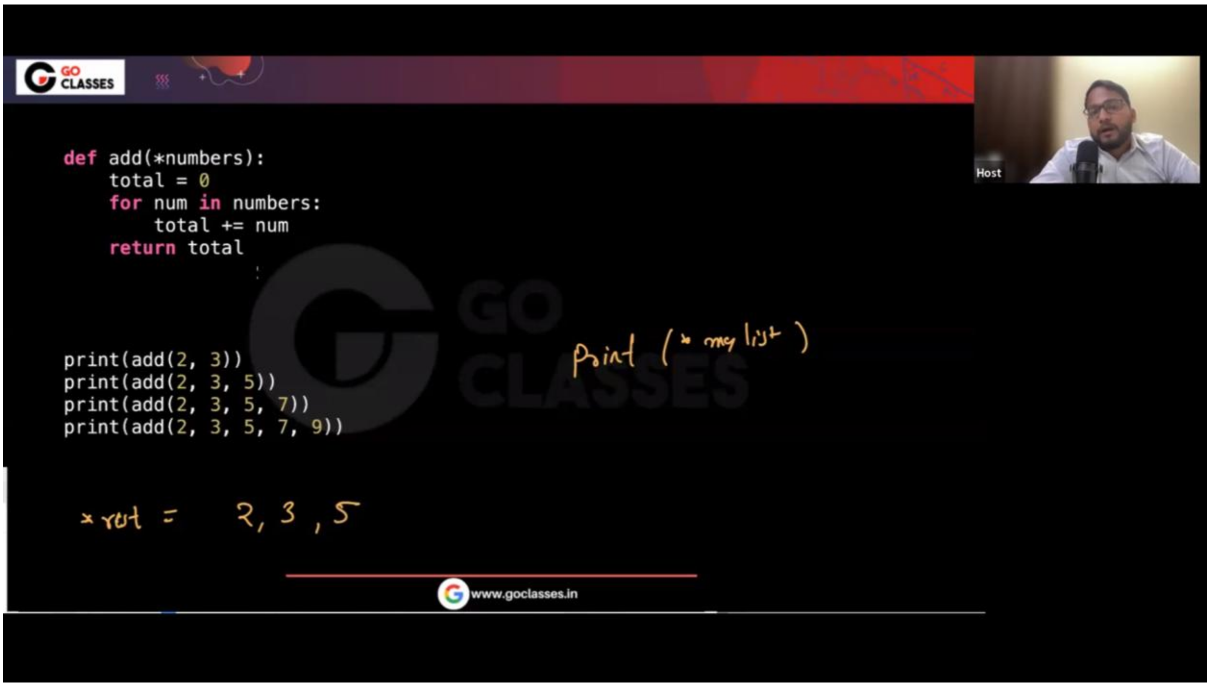
```
In [7]: my_list = [1,2,3,4]

        a,b,c,d,e = my_list

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

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
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)
```


localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb
Host
1:08:40
-18:08

UPDATE 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 might break some of your extensions.

Jupyter Python Programming GO Classes Last Checkpoint: Yesterday at 3:00 PM (unsaved changes)

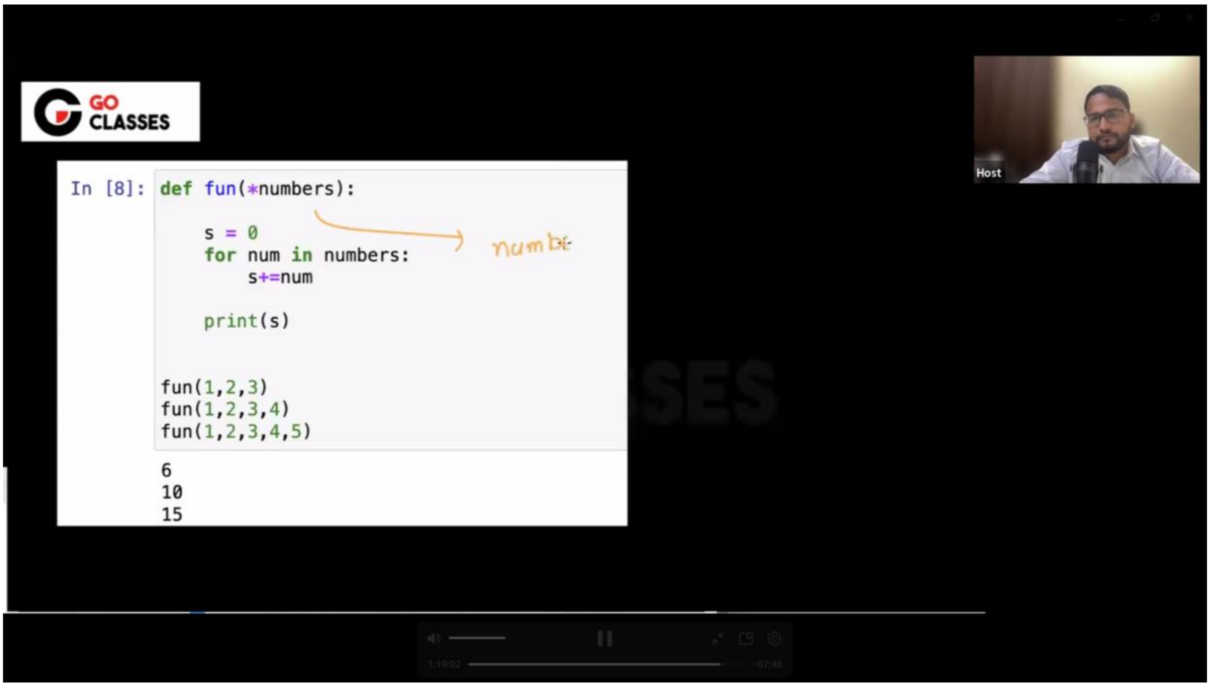
```
In [11]: numbers = [2,1,3,4,7]

         print(*numbers)

2 1 3 4 7

In [10]: print(numbers[0],numbers[1],numbers[2],numbers[3],numbers[4])

2 1 3 4 7

In [ ]:
```



```
def add(*numbers):
    total = 0
    for num in numbers:
        total += num
    return total


print(add(2, 3))
print(add(2, 3, 5))
print(add(2, 3, 5, 7))
print(add(2, 3, 5, 7, 9))
```

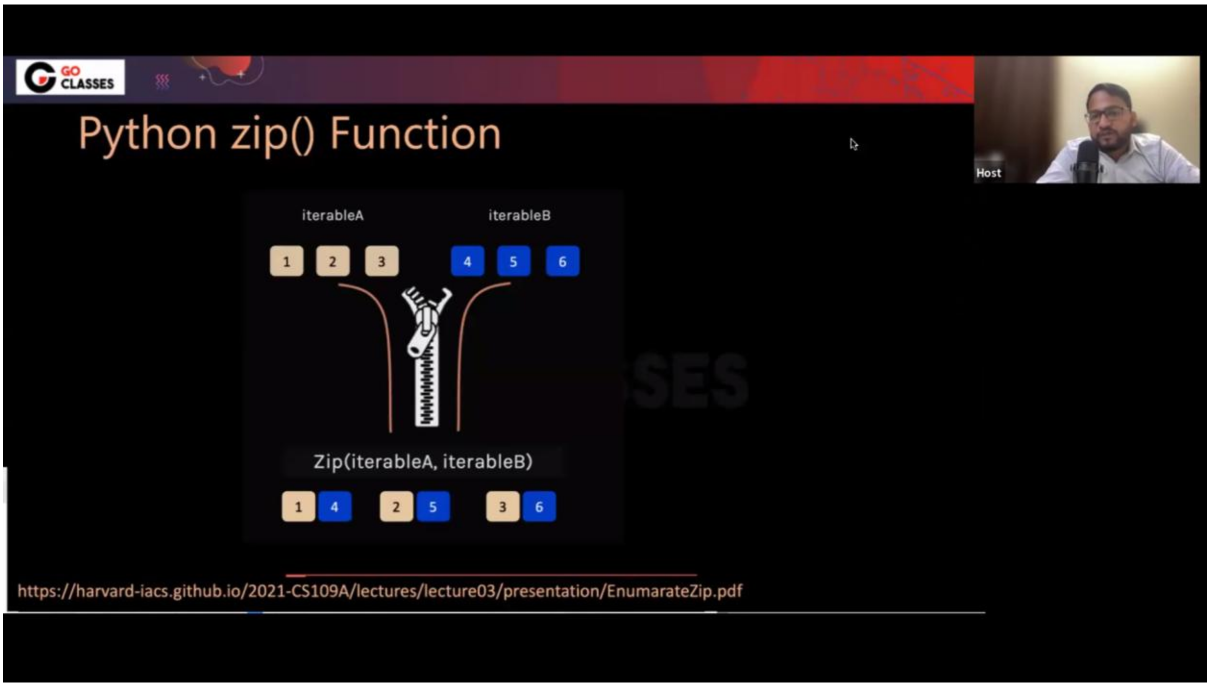
Print (* my list)

* rest = 2, 3, 5



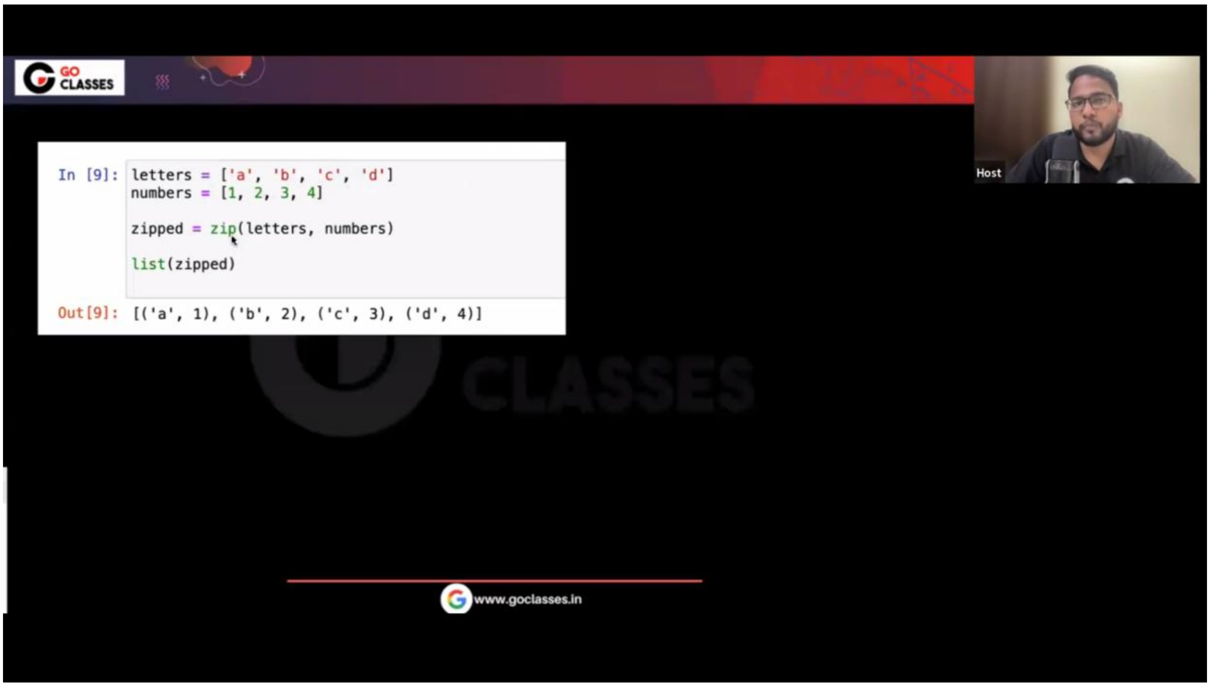
```
In [8]: def fun(*numbers):

            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
```

numbe

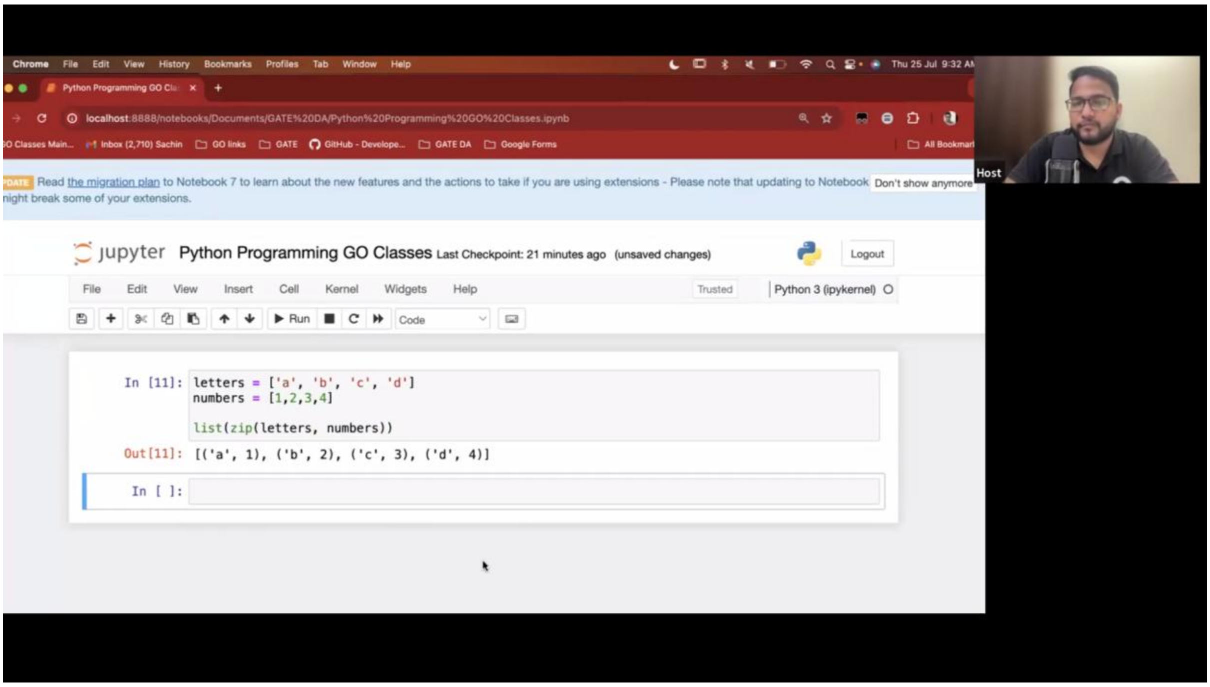


Python zip() Function

https://harvard-iacs.github.io/2021-CS109A/lectures/lecture03/presentation/EnumarateZip.pdf



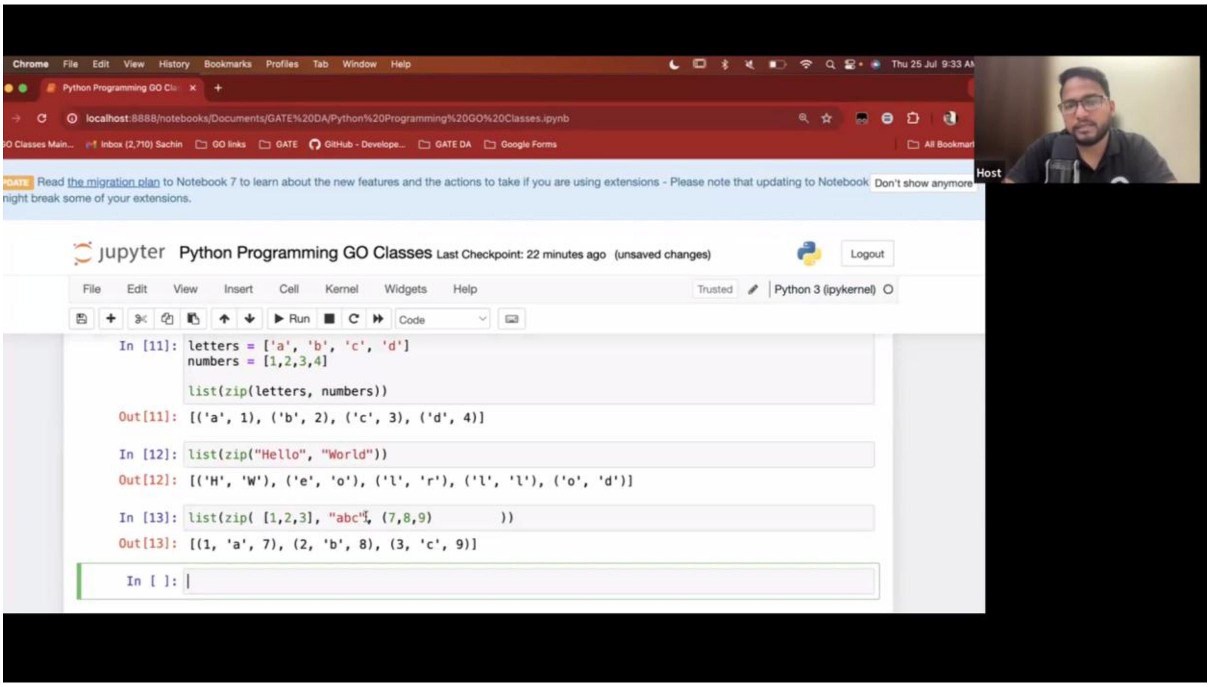
```
In [9]: letters = ['a', 'b', 'c', 'd']
        numbers = [1, 2, 3, 4]

        zipped = zip(letters, numbers)

        list(zipped)

Out[9]: [('a', 1), ('b', 2), ('c', 3), ('d', 4)]
```



Chrome File Edit View History Bookmarks Profiles Tab Window Help
Thu 25 Jul 9:32 AM
Python Programming GO Cla
localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb
O Classes Main... Inbox (2,710) Sachin GO links GATE GitHub - Develope... GATE DA Google Forms All Bookmarks
Host
jupyter Python Programming GO Classes Last Checkpoint: 21 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

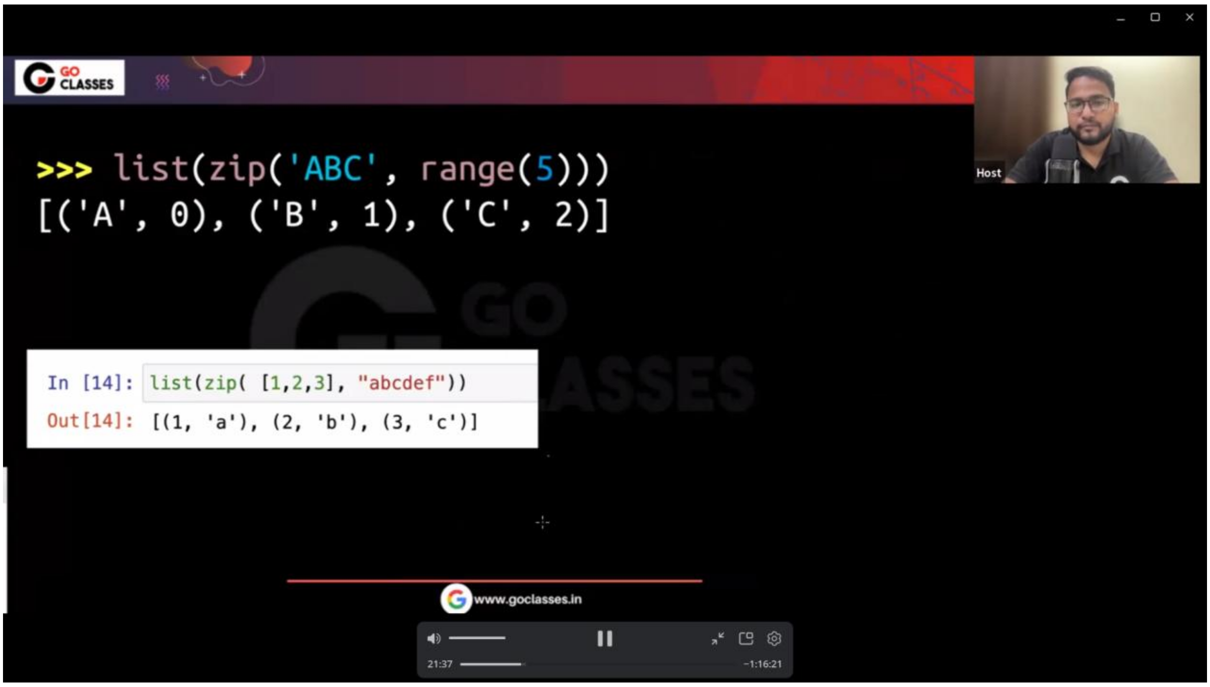
PDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook might break some of your extensions. Don't show anymore

```
In [11]: letters = ['a', 'b', 'c', 'd']
         numbers = [1,2,3,4]

         list(zip(letters, numbers))

Out[11]: [('a', 1), ('b', 2), ('c', 3), ('d', 4)]

In [ ]:
```



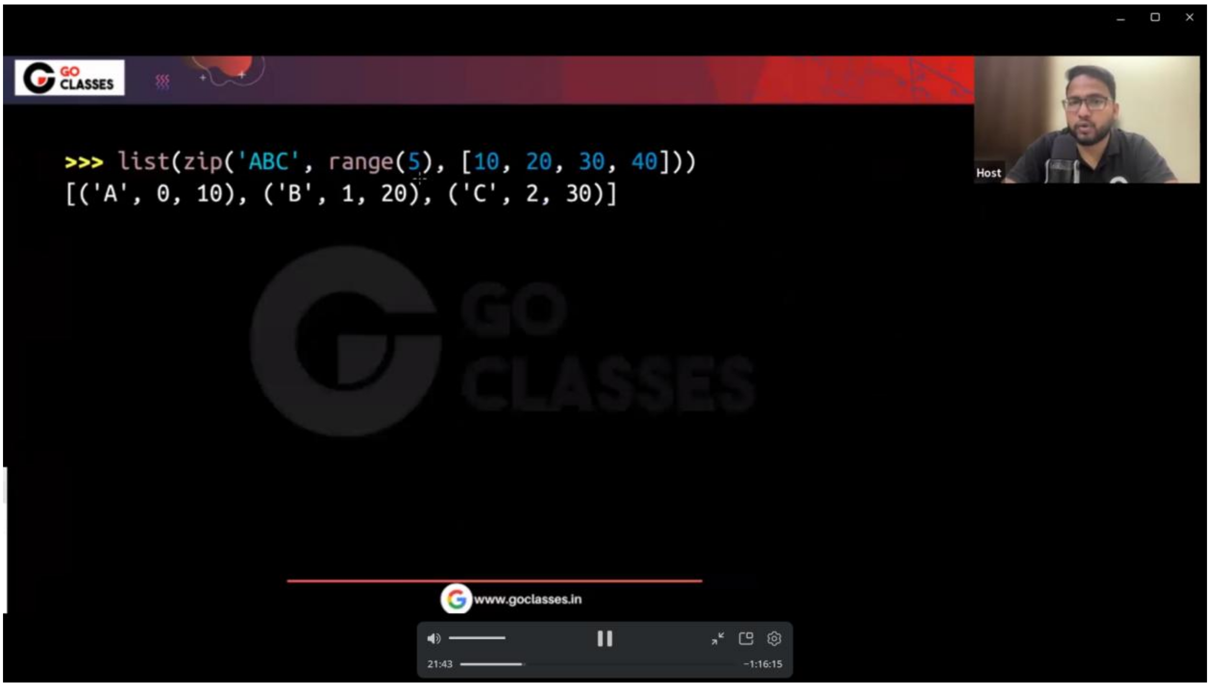
Chrome File Edit View History Bookmarks Profiles Tab Window Help
Thu 25 Jul 9:33 AM
Python Programming GO Cla
localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb
Host
Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook might break some of your extensions.
Don't show anymore
jupyter Python Programming GO Classes Last Checkpoint: 22 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [11]: letters = ['a', 'b', 'c', 'd']
         numbers = [1,2,3,4]

         list(zip(letters, numbers))

Out[11]: [('a', 1), ('b', 2), ('c', 3), ('d', 4)]

In [12]: list(zip("Hello", "World"))

Out[12]: [('H', 'W'), ('e', 'o'), ('l', 'r'), ('l', 'l'), ('o', 'd')]

In [13]: list(zip( [1,2,3], "abc", (7,8,9)          ))

Out[13]: [(1, 'a', 7), (2, 'b', 8), (3, 'c', 9)]

In [ ]:
```

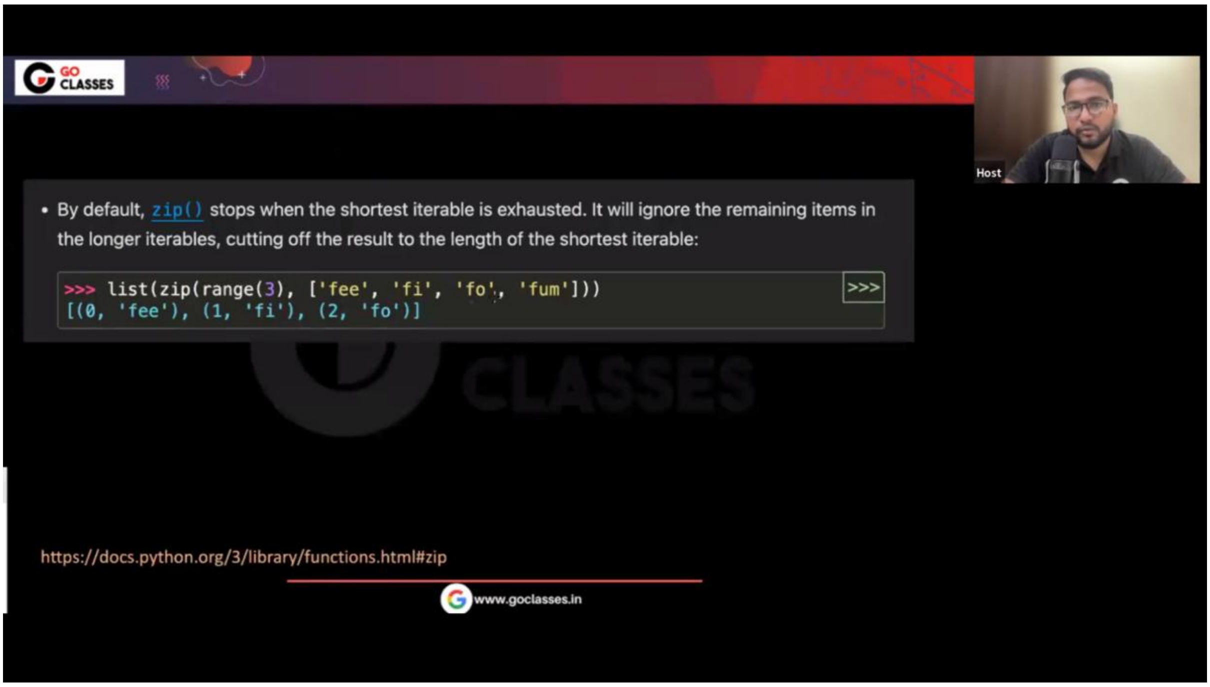


```
>>> list(zip('ABC', range(5)))
[('A', 0), ('B', 1), ('C', 2)]
```

```
In [14]: list(zip( [1,2,3], "abcdef"))
Out[14]: [(1, 'a'), (2, 'b'), (3, 'c')]
```



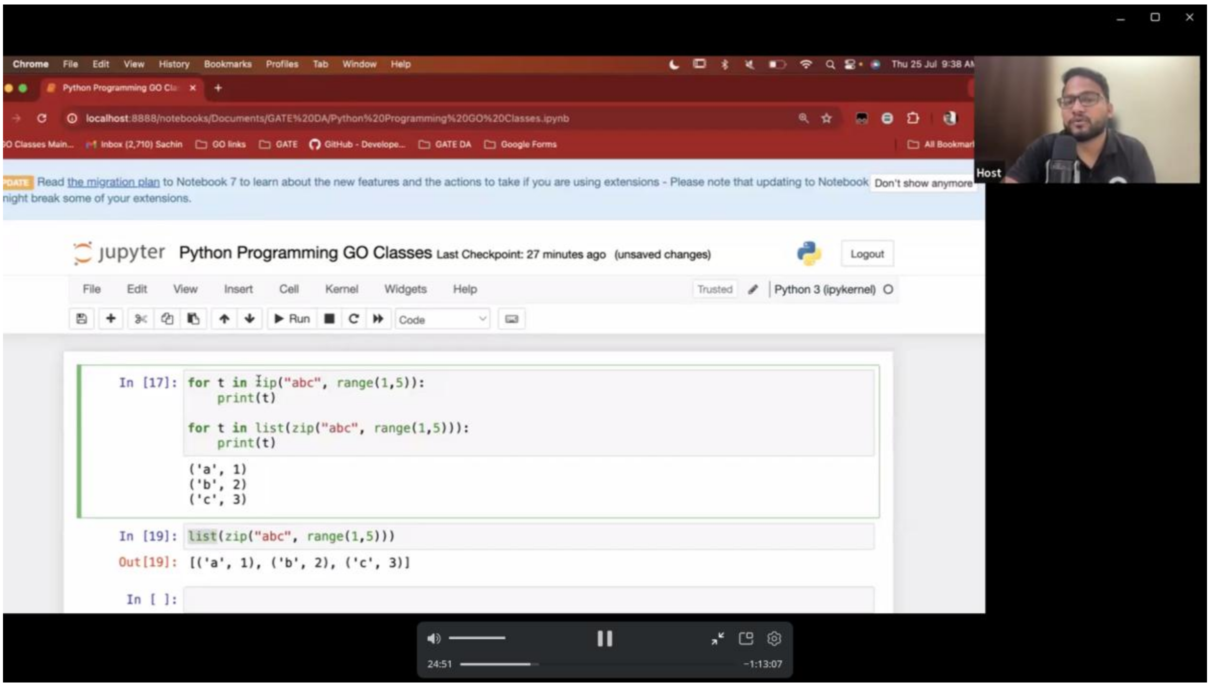
```
>>> list(zip('ABC', range(5), [10, 20, 30, 40]))
[('A', 0, 10), ('B', 1, 20), ('C', 2, 30)]
```



- 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')]
```

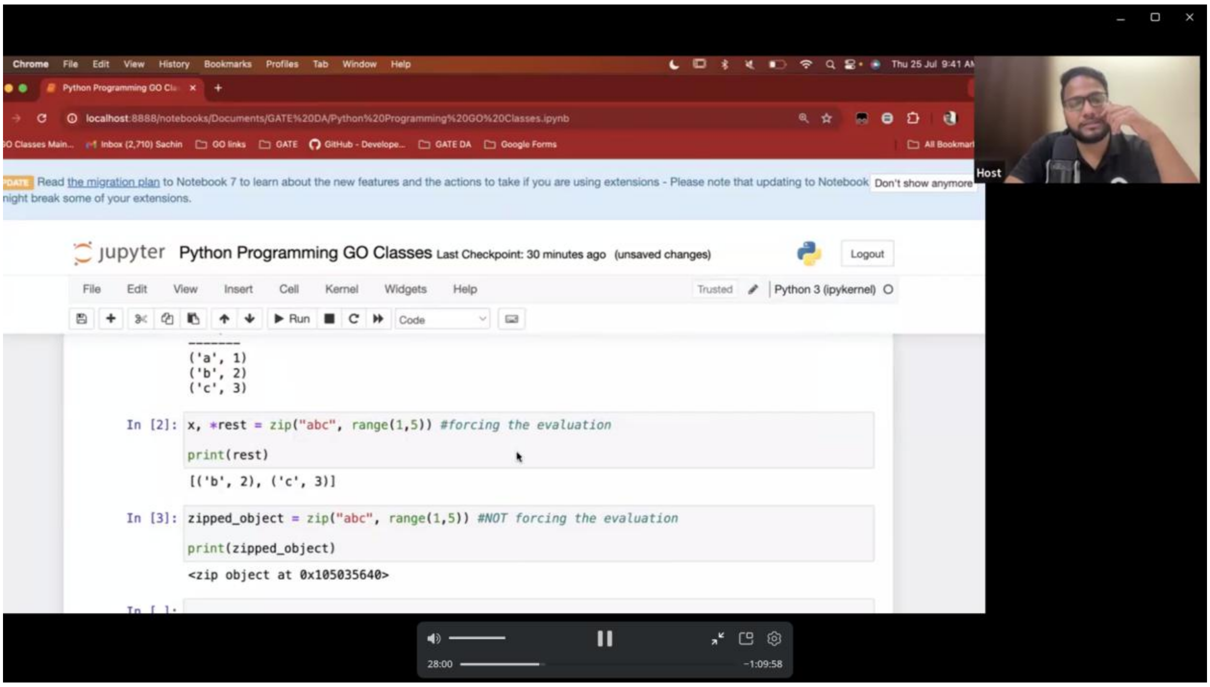
https://docs.python.org/3/library/functions.html#zip


localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb
Host
24:51
-1:13:07

PDATE 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 night break some of your extensions.

Jupyter Python Programming GO Classes Last Checkpoint: 27 minutes ago (unsaved changes)

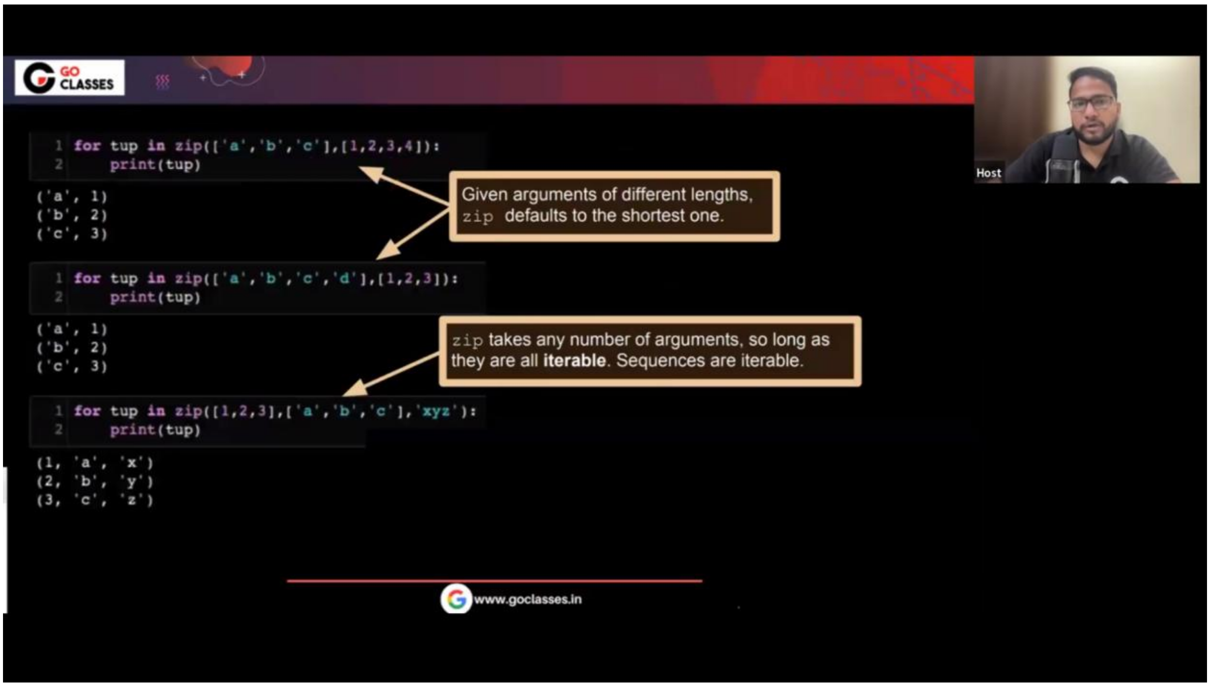
```
In [17]: for t in zip("abc", range(1,5)):
             print(t)

         for t in list(zip("abc", range(1,5))):
             print(t)

         ('a', 1)
         ('b', 2)
         ('c', 3)

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

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

In [ ]:
```


localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb
Host
28:00
-1:09:58

PDATE 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 night break some of your extensions.

Jupyter Python Programming GO Classes Last Checkpoint: 30 minutes ago (unsaved changes)

```
         -------
         ('a', 1)
         ('b', 2)
         ('c', 3)

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

        print(rest)

        [('b', 2), ('c', 3)]

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

        print(zipped_object)

        <zip object at 0x105035640>

In [ ]:
```



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

```
('a', 1)
('b', 2)
('c', 3)
```

Given arguments of different lengths, `zip` defaults to the shortest one.

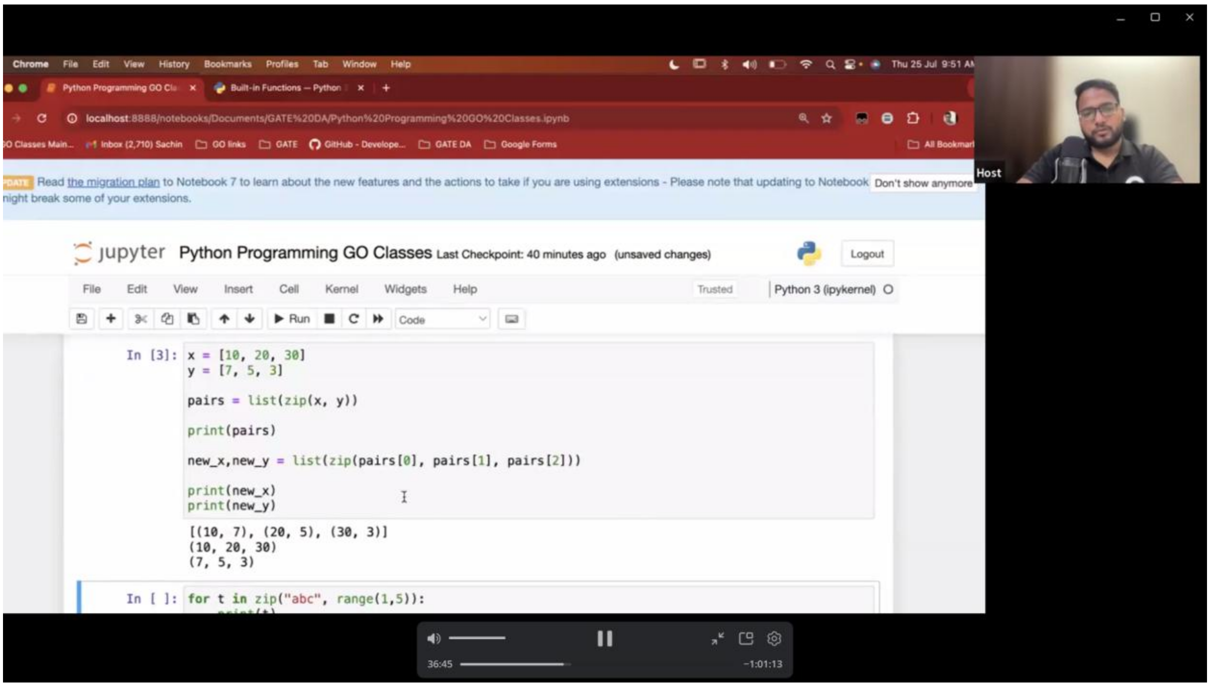
```
for tup in zip(['a','b','c','d'],[1,2,3]):
    print(tup)
```

```
('a', 1)
('b', 2)
('c', 3)
```

`zip` takes any number of arguments, so long as they are all **iterable**. Sequences are iterable.

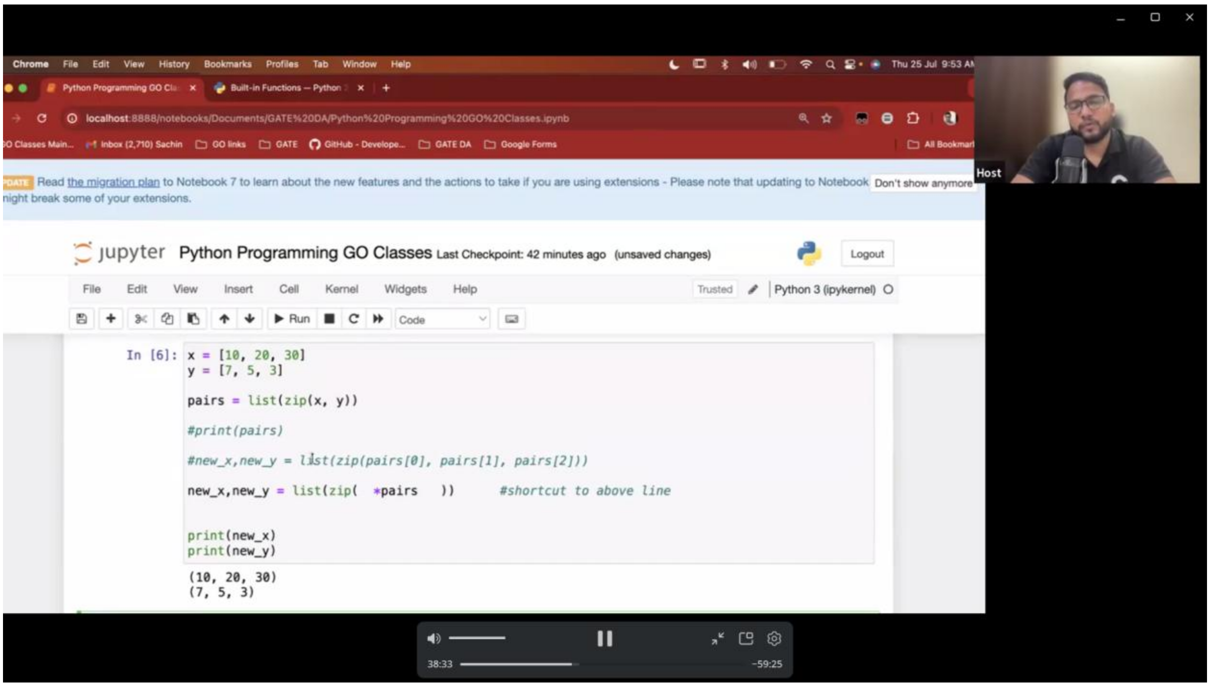
```
for tup in zip([1,2,3],['a','b','c'],'xyz'):
    print(tup)
```

```
(1, 'a', 'x')
(2, 'b', 'y')
(3, 'c', 'z')
```

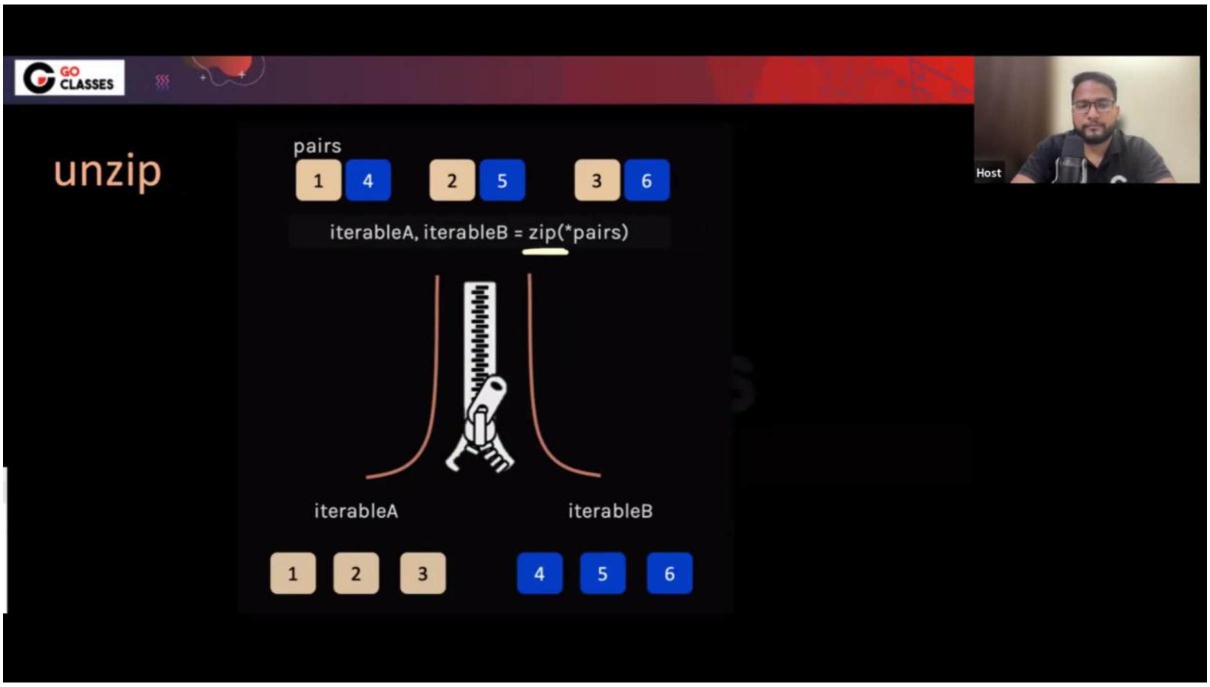


jupyter Python Programming GO Classes Last Checkpoint: 40 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Host
36:45
-1:01:13

```
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)

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



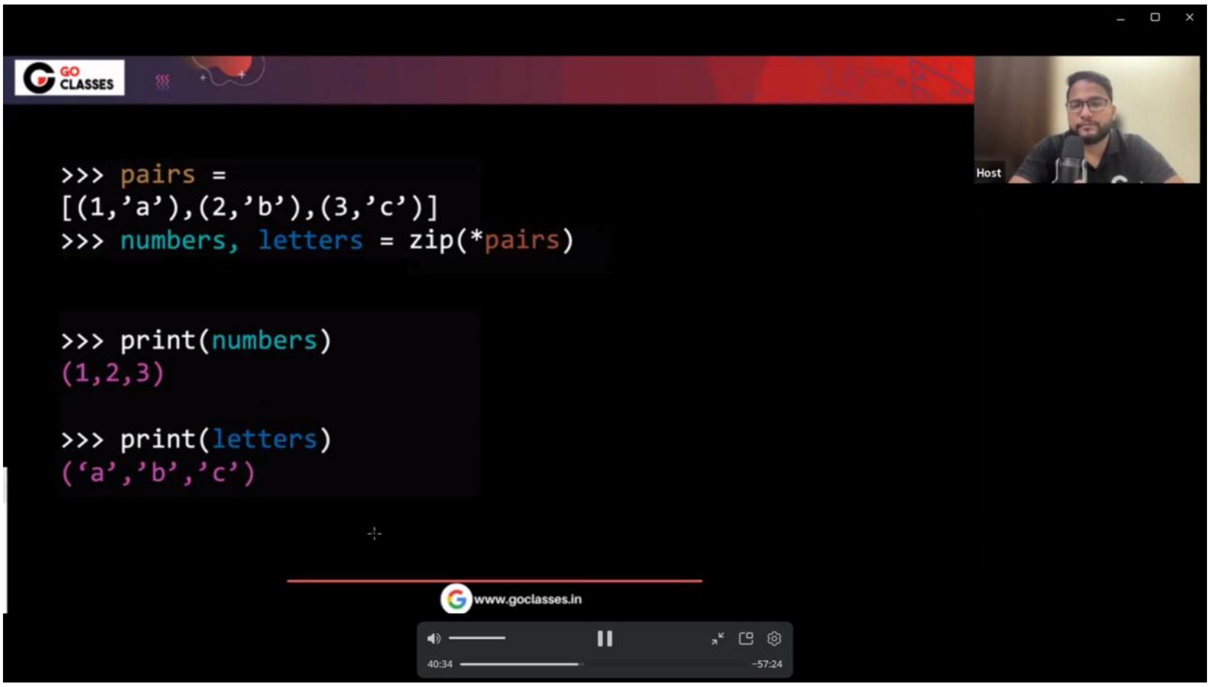
```
In [6]: x = [10, 20, 30]
        y = [7, 5, 3]

        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)
```



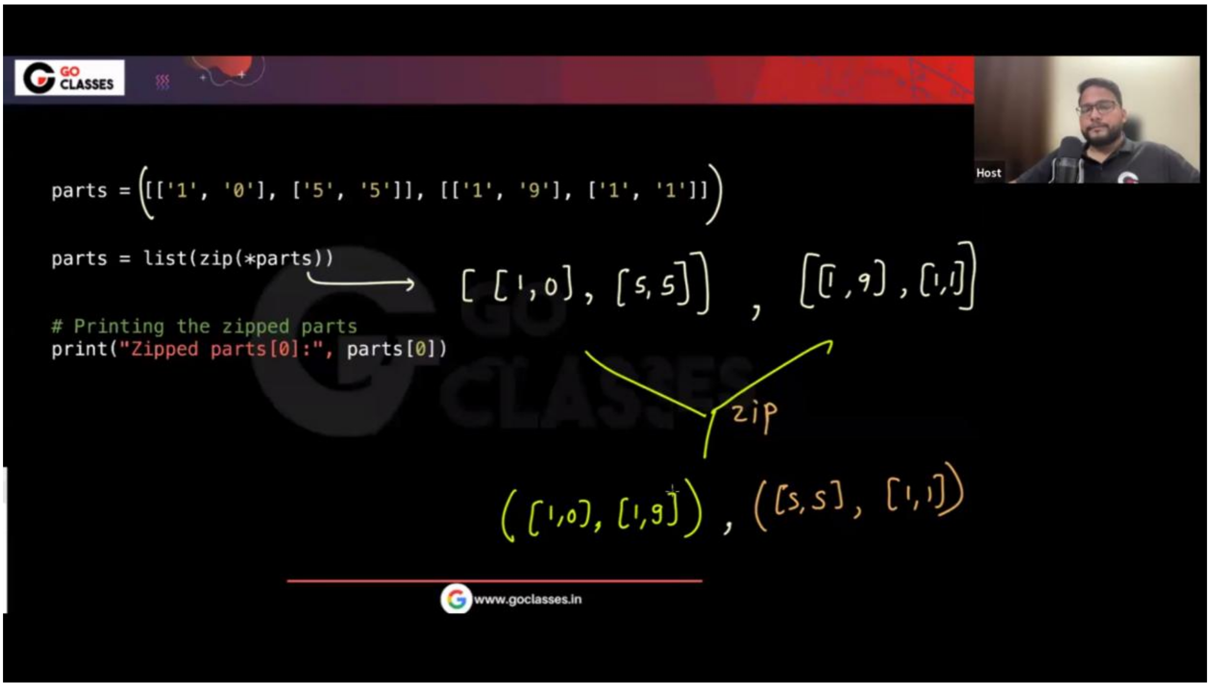
unzip

pairs

iterableA, iterableB = zip(*pairs)



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

>>> print(numbers)
(1,2,3)

>>> print(letters)
('a','b','c')
```

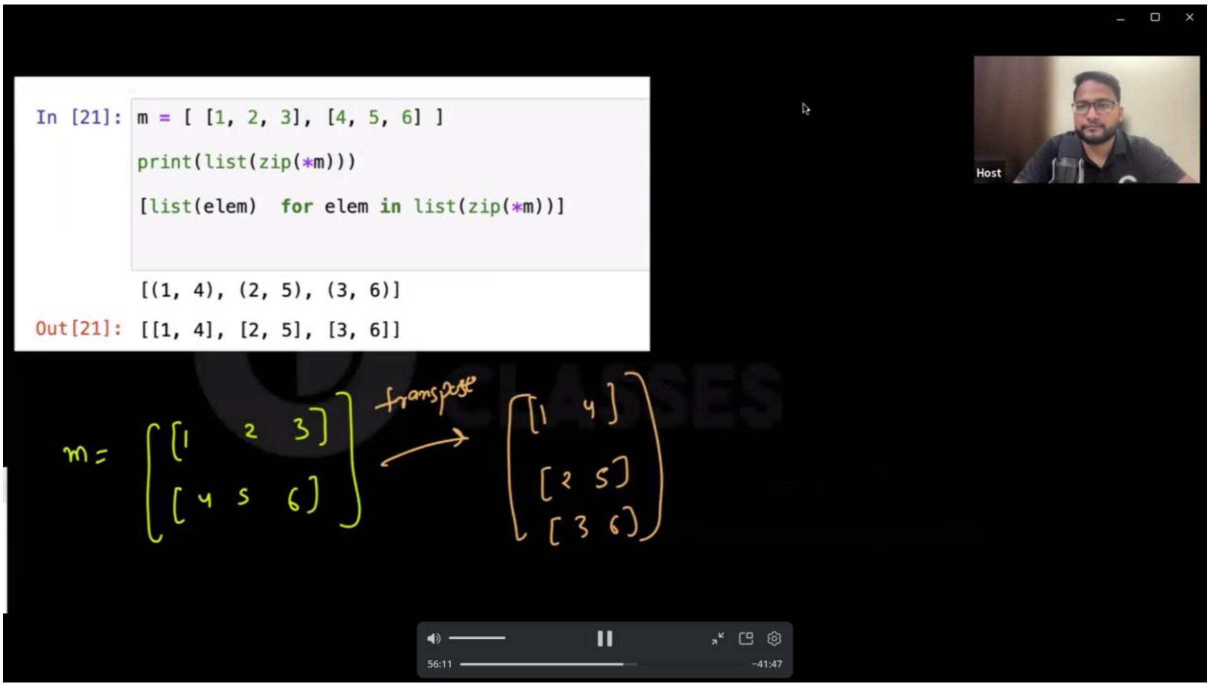


```
parts = ([['1', '0'], ['5', '5']], [['1', '9'], ['1', '1']])

parts = list(zip(*parts))

# Printing the zipped parts
print("Zipped parts[0]:", parts[0])
```

[[1,0], [5,5]] , [[1,9], [1,1]]

zip

([1,0], [1,9]) , ([5,5], [1,1])

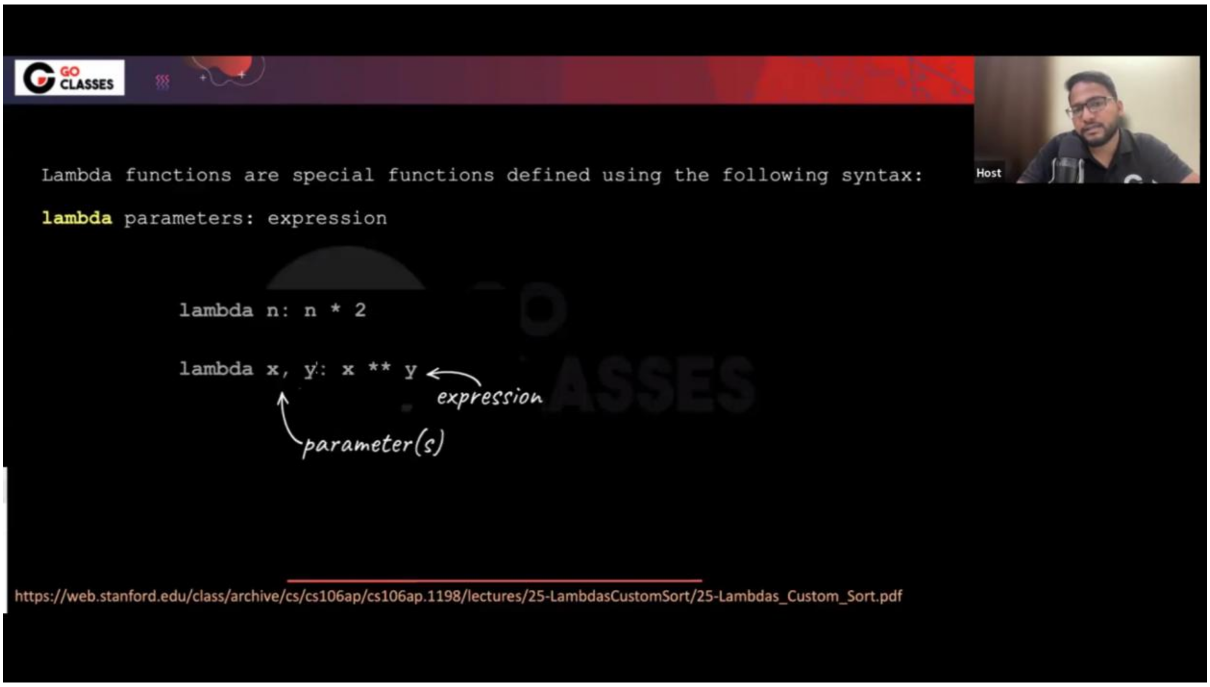


```
In [21]: m = [ [1, 2, 3], [4, 5, 6] ]

         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]]
```

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

Next Topic:

Lambda function

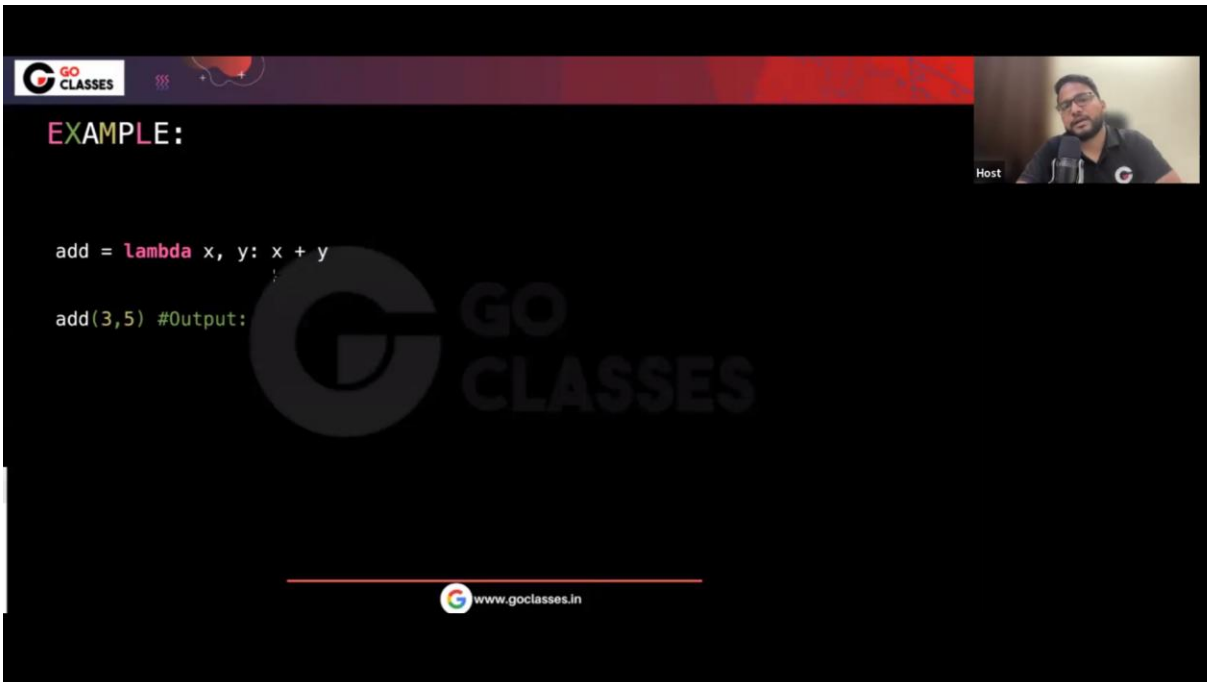


Lambda functions are special functions defined using the following syntax:

```
lambda parameters: expression
```

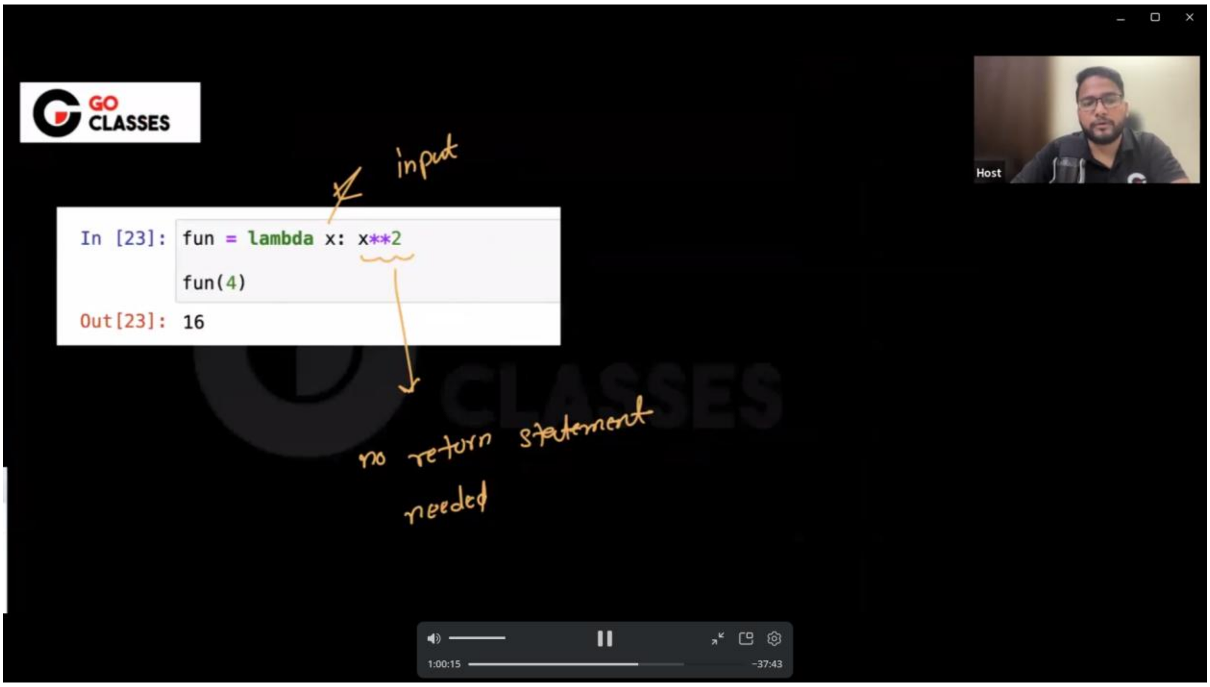
```
lambda n: n * 2

lambda x, y: x ** y
```

https://web.stanford.edu/class/archive/cs/cs106ap/cs106ap.1198/lectures/25-LambdasCustomSort/25-Lambdas_Custom_Sort.pdf



EXAMPLE:

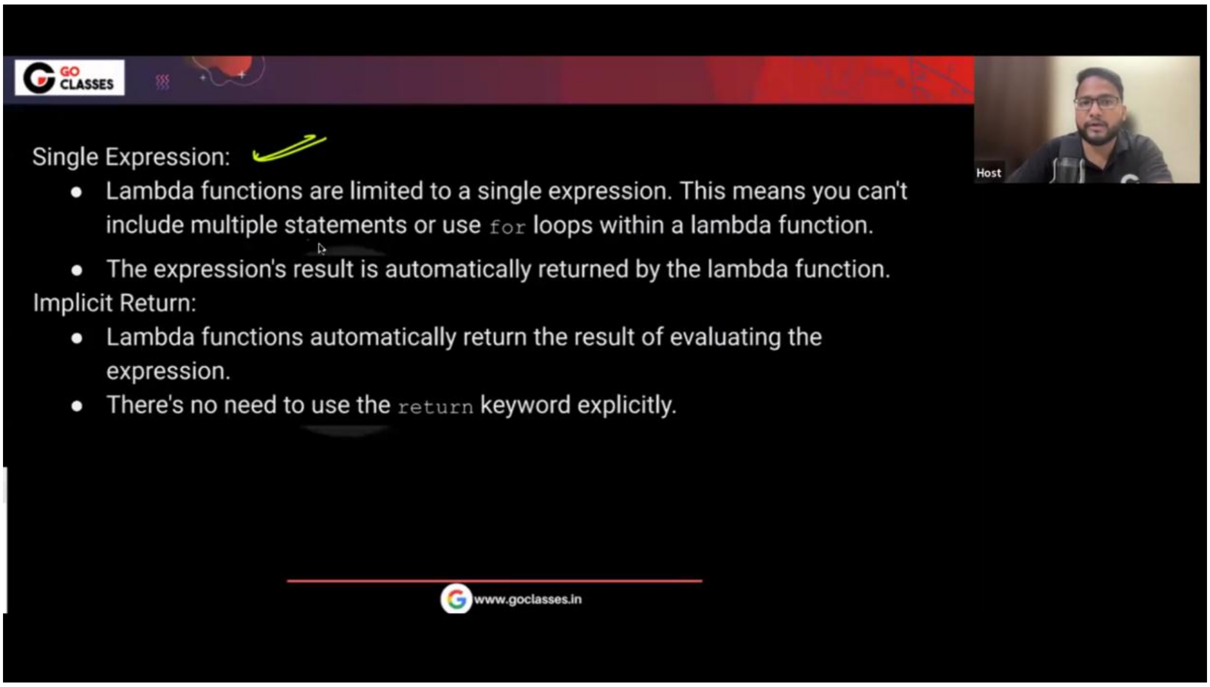
```
add = lambda x, y: x + y

add(3,5) #Output:
```



```
In [23]: fun = lambda x: x**2

         fun(4)

Out[23]: 16
```

input

no return statement needed

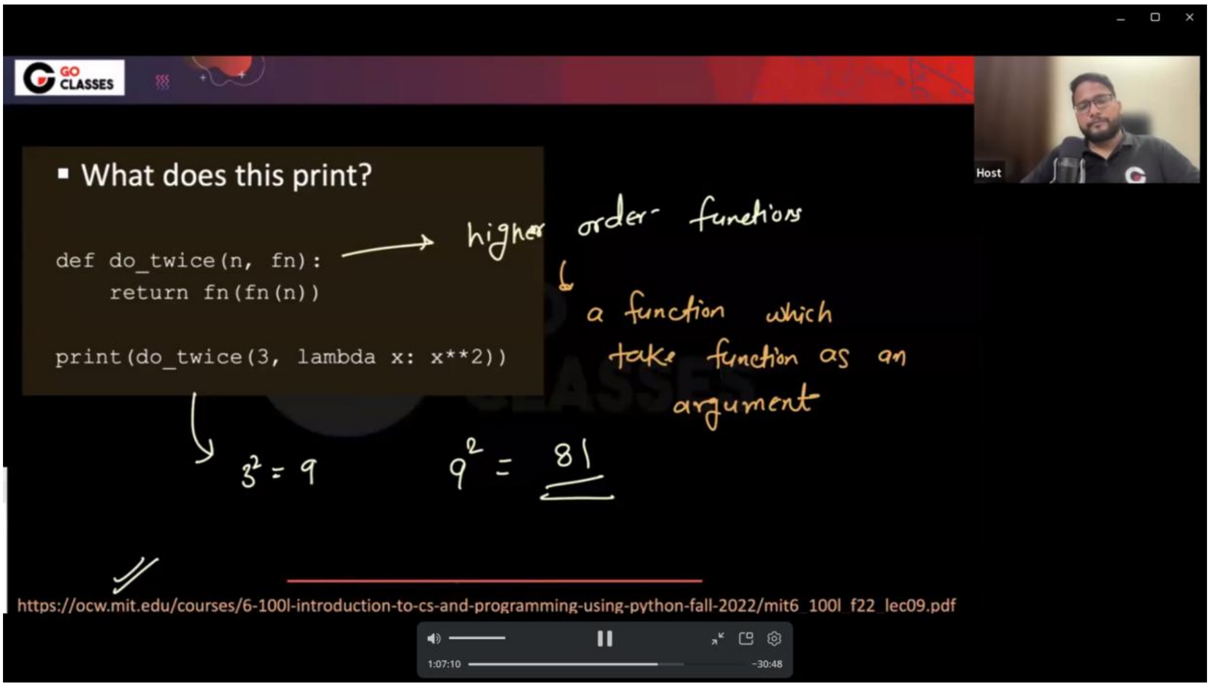


Single Expression:

- 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.

Implicit Return:

- Lambda functions automatically return the result of evaluating the expression.
- There's no need to use the `return` keyword explicitly.



- What does this print?

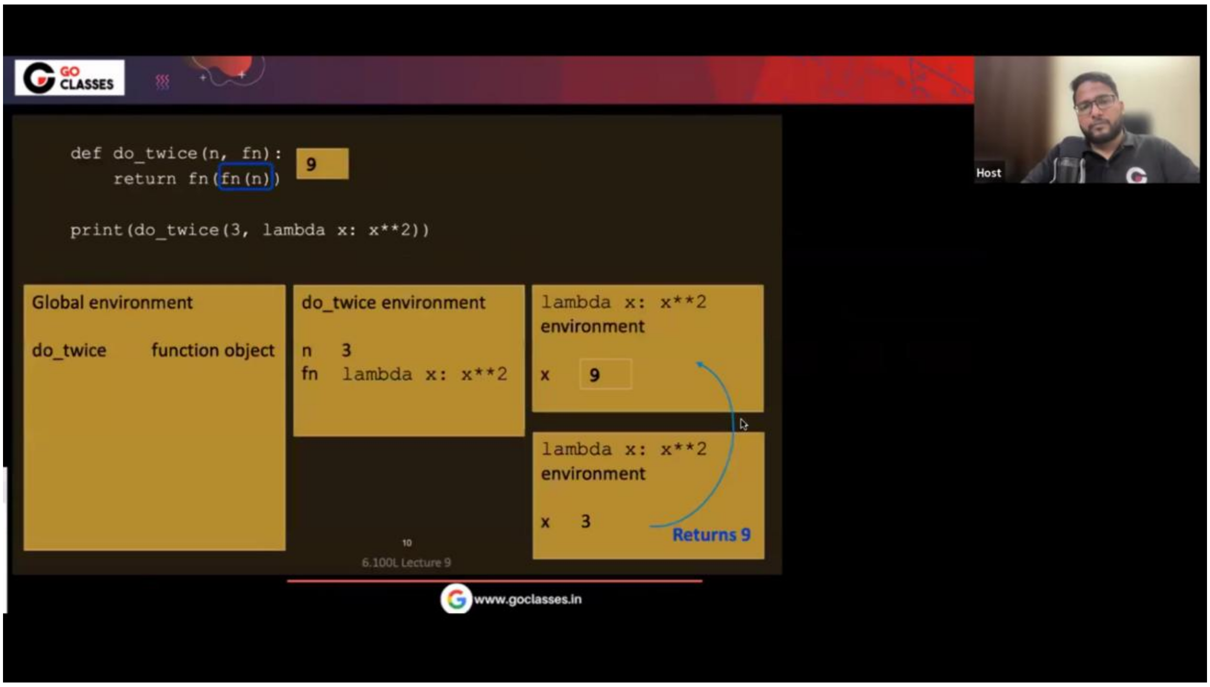
```
def do_twice(n, fn):
    return fn(fn(n))

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

→ higher order functions

a function which take function as an argument

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

https://ocw.mit.edu/courses/6-100l-introduction-to-cs-and-programming-using-python-fall-2022/mit6_100l_f22_lec09.pdf



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

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

9

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

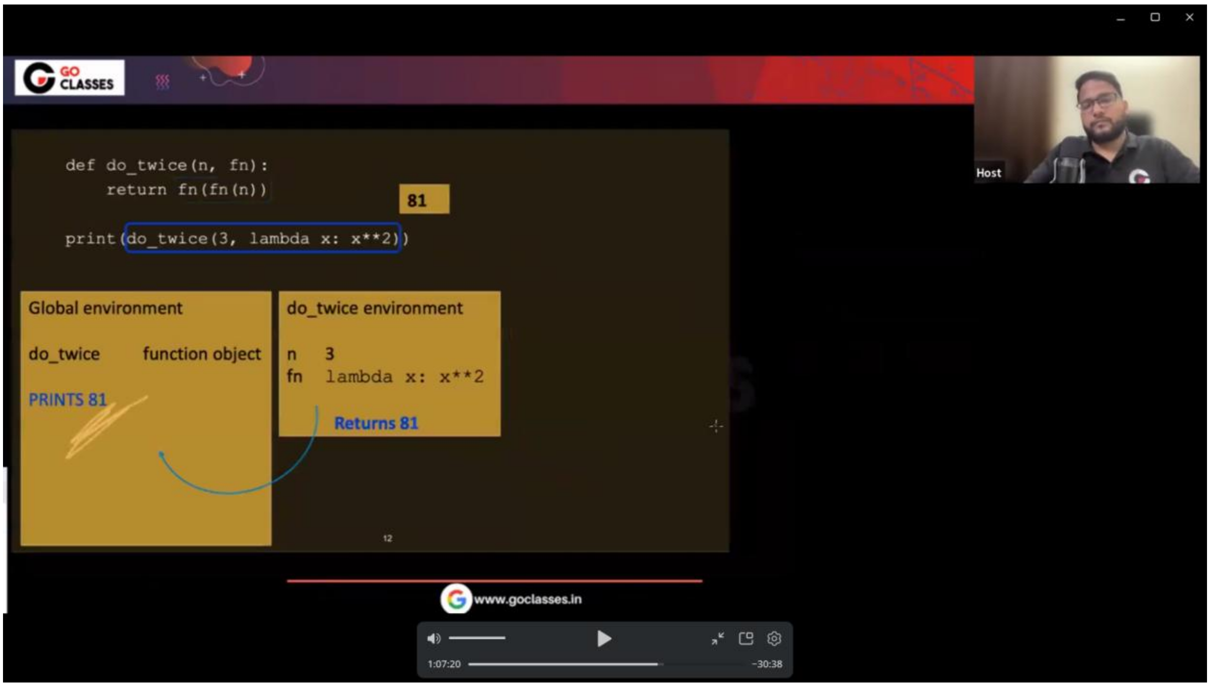
x 3

Returns 9

10

6.100L Lecture 9

www.goclasses.in



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

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

81

Global environment

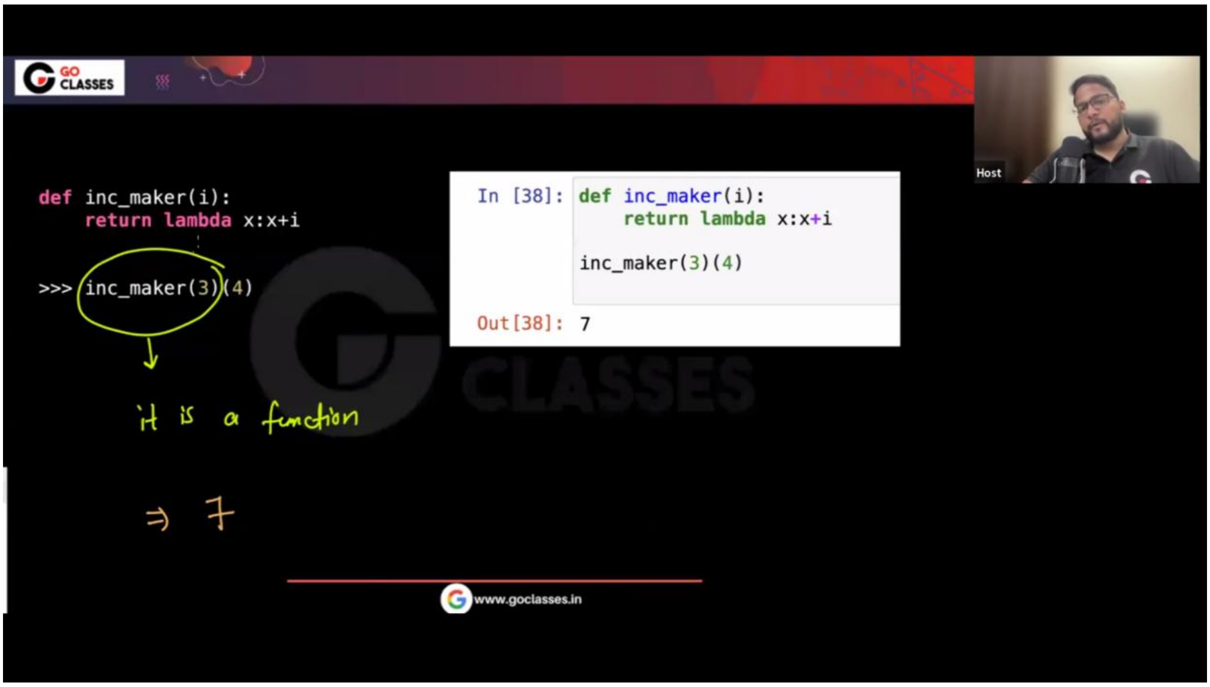
do_twice function object

PRINTS 81

do_twice environment

n 3
fn lambda x: x**2

Returns 81

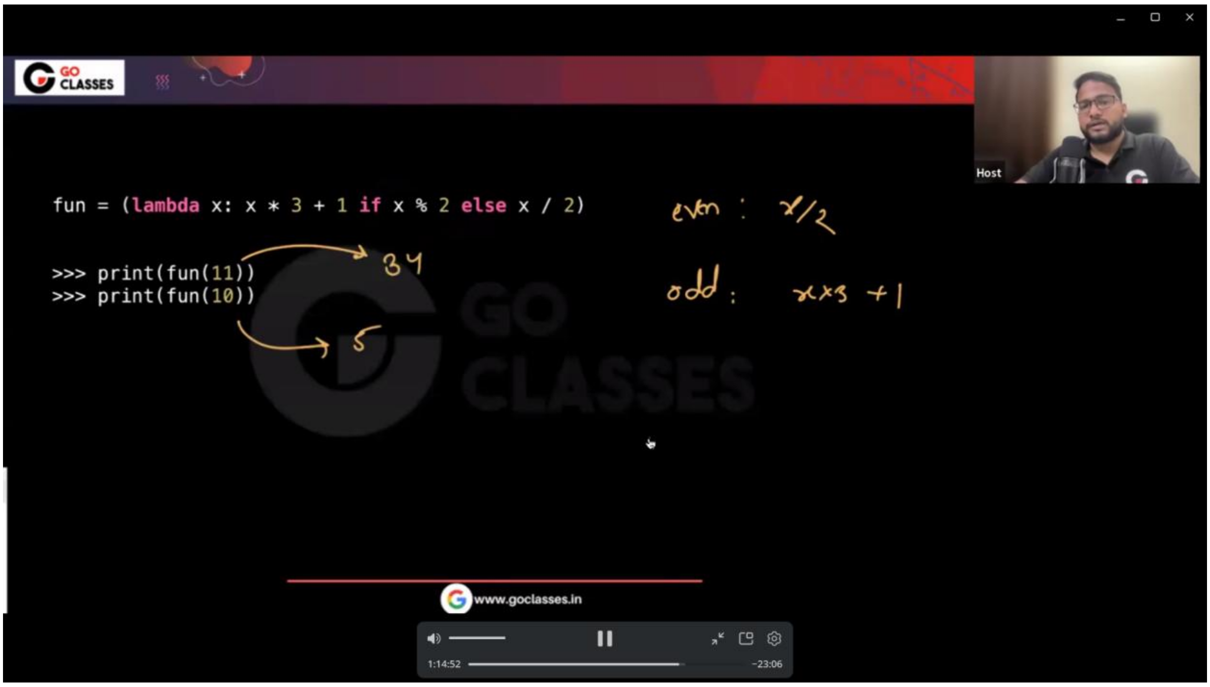


```
def inc_maker(i):
    return lambda x:x+i

>>> inc_maker(3)(4)
```

it is a function

⇒ 7

```
In [38]: def inc_maker(i):
             return lambda x:x+i

         inc_maker(3)(4)

Out[38]: 7
```



```
fun = (lambda x: x * 3 + 1 if x % 2 else x / 2)

>>> print(fun(11))
>>> print(fun(10))
```

34

5

even : $x/2$

odd : $x \times 3 + 1$