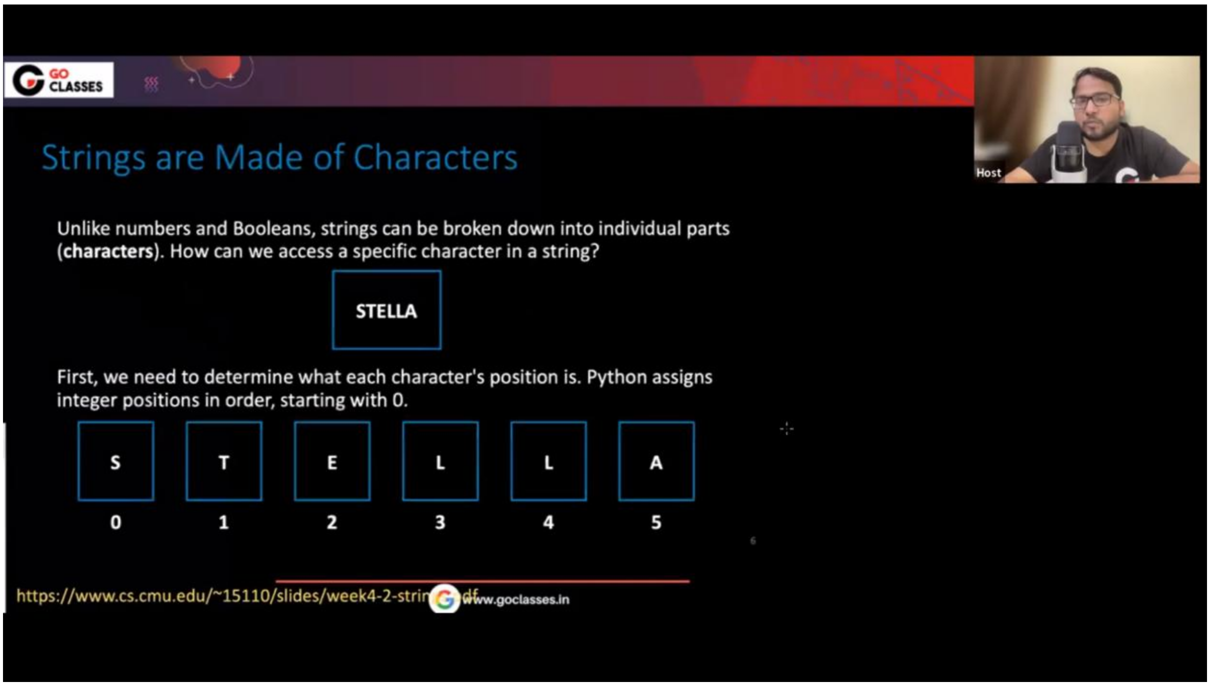


Strings are Made of Characters

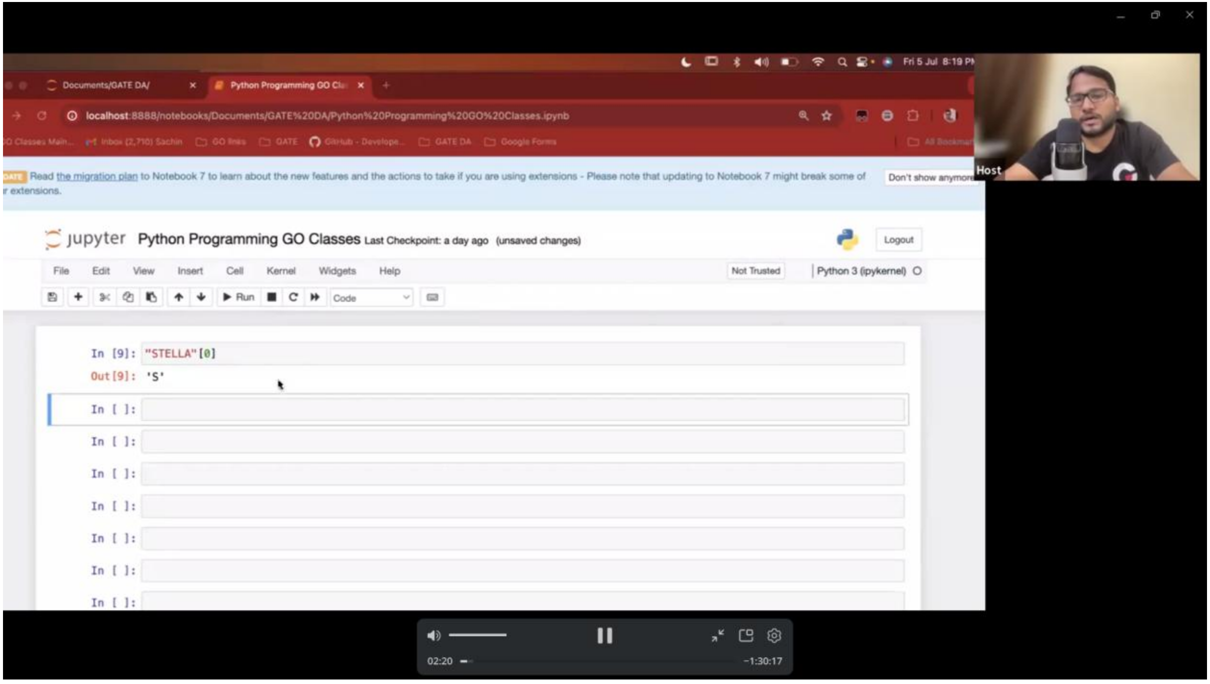
Unlike numbers and Booleans, strings can be broken down into individual parts (**characters**). How can we access a specific character in a string?

STELLA

First, we need to determine what each character's position is. Python assigns integer positions in order, starting with 0.

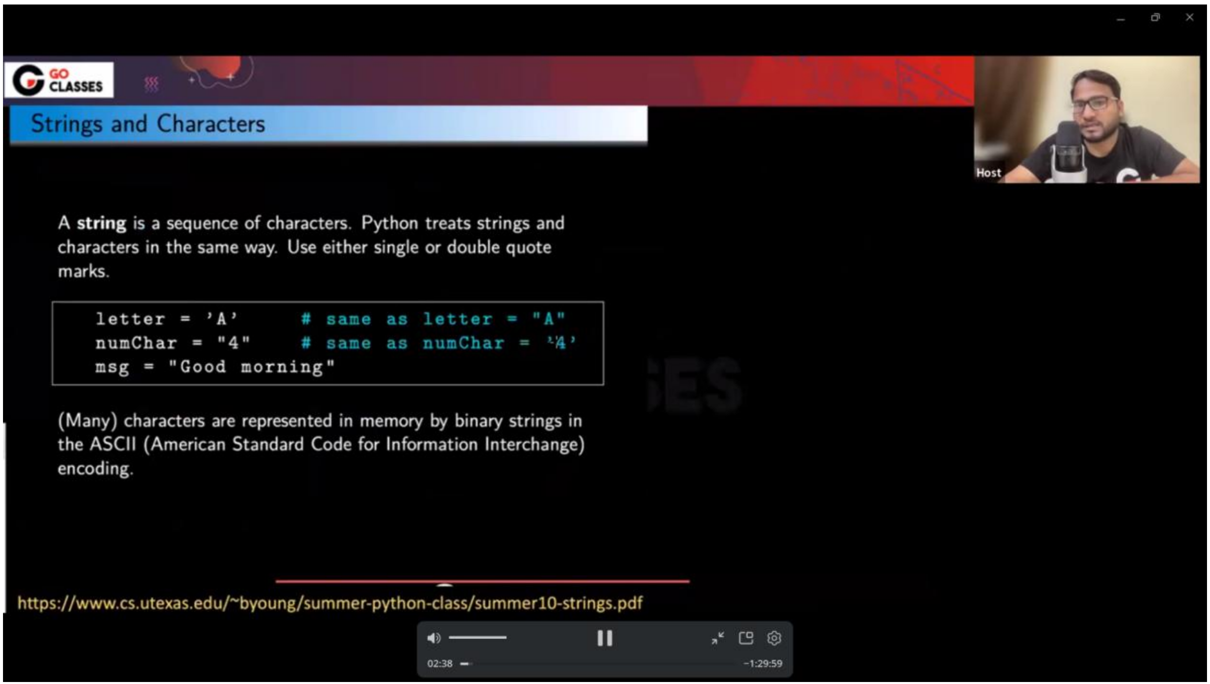
S	T	E	L	L	A
0	1	2	3	4	5

https://www.cs.cmu.edu/~15110/slides/week4-2-strin...df www.goclasses.in



Fri 5 Jul 8:19 PM
Documents/GATE DA/
Python Programming GO Cla
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 r extensions.
Don't show anymore
jupyter
Python Programming GO Classes Last Checkpoint: a day ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Host
02:20
-1:30:17

```
In [9]: "STELLA"[0]
Out[9]: 'S'

In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
```



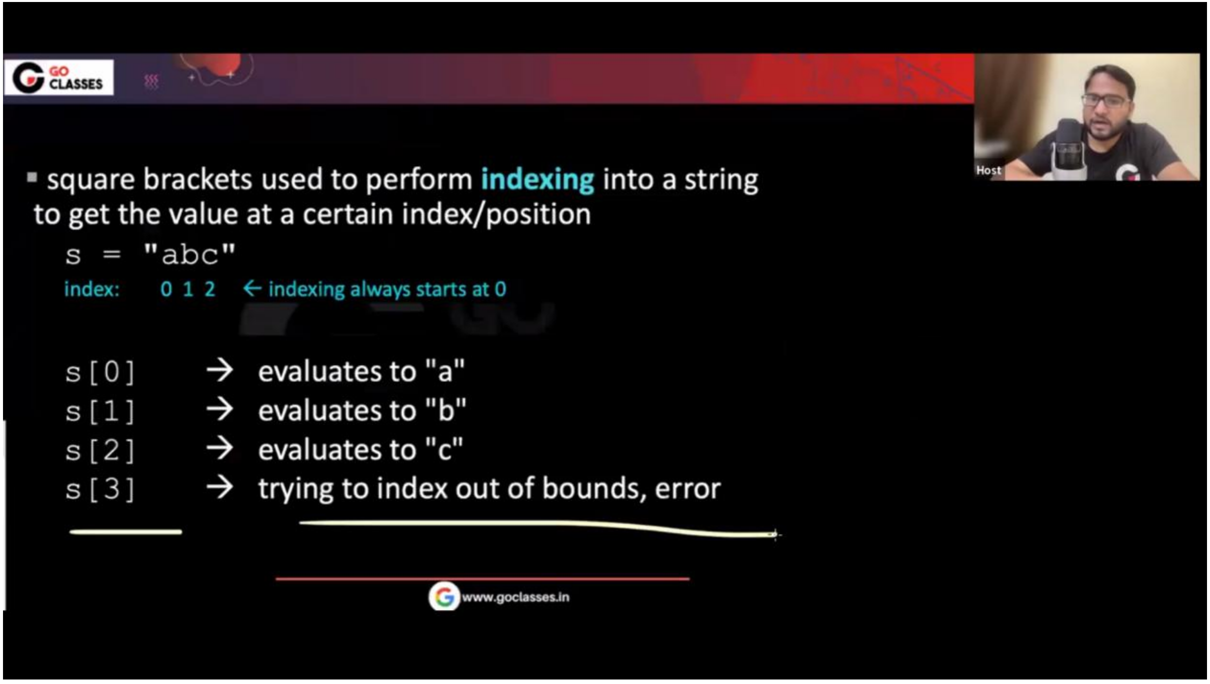
Strings and Characters

A **string** is a sequence of characters. Python treats strings and characters in the same way. Use either single or double quote marks.

```
letter = 'A'      # same as letter = "A"
numChar = "4"     # same as numChar = '4'
msg = "Good morning"
```

(Many) characters are represented in memory by binary strings in the ASCII (American Standard Code for Information Interchange) encoding.

https://www.cs.utexas.edu/~byoung/summer-python-class/summer10-strings.pdf



- square brackets used to perform **indexing** into a string to get the value at a certain index/position

```
s = "abc"
```

index: 0 1 2 ← indexing always starts at 0

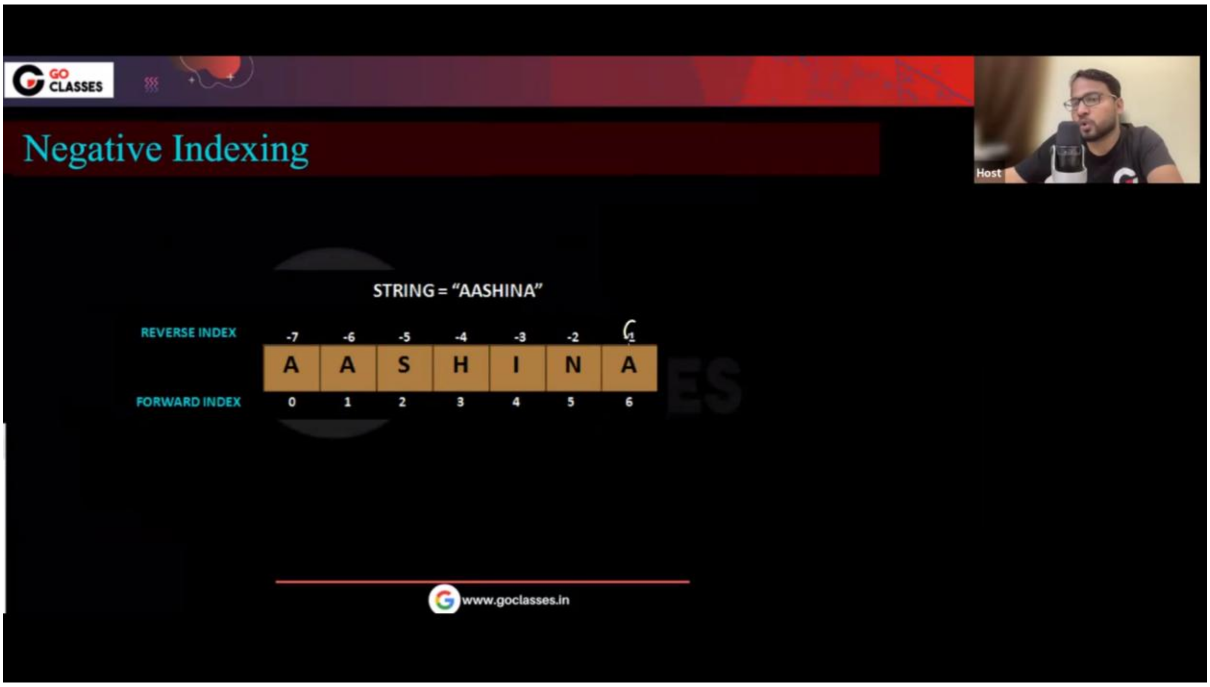
s[0] → evaluates to "a"

s[1] → evaluates to "b"

s[2] → evaluates to "c"

s[3] → trying to index out of bounds, error

www.goclasses.in

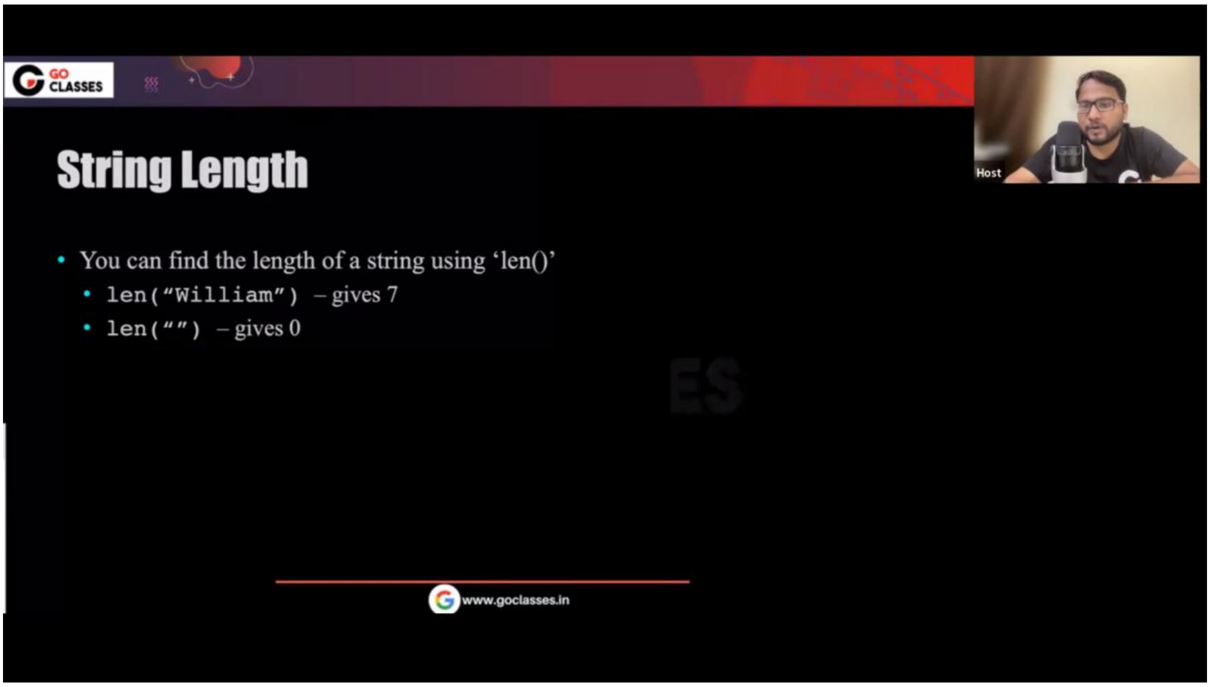


Negative Indexing

STRING = "AASHINA"

REVERSE INDEX	-7	-6	-5	-4	-3	-2	-1
	A	A	S	H	I	N	A
FORWARD INDEX	0	1	2	3	4	5	6

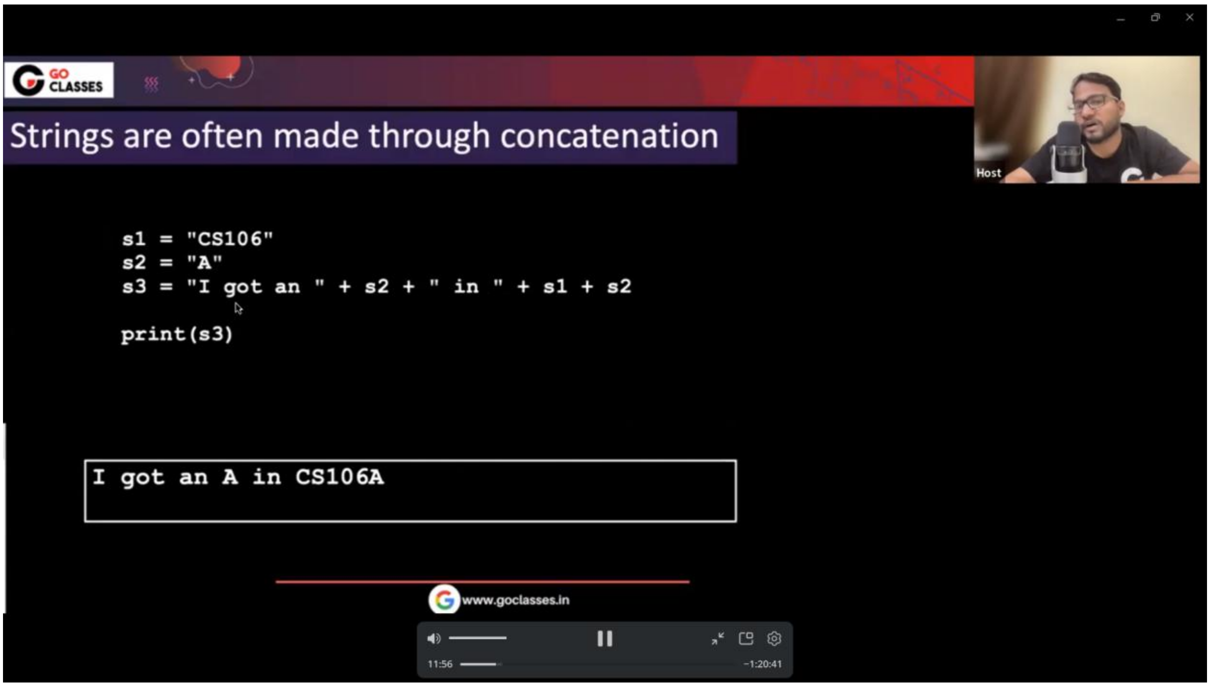
www.goclasses.in



String Length

- You can find the length of a string using 'len()'
  - `len("William")` – gives 7
  - `len("")` – gives 0

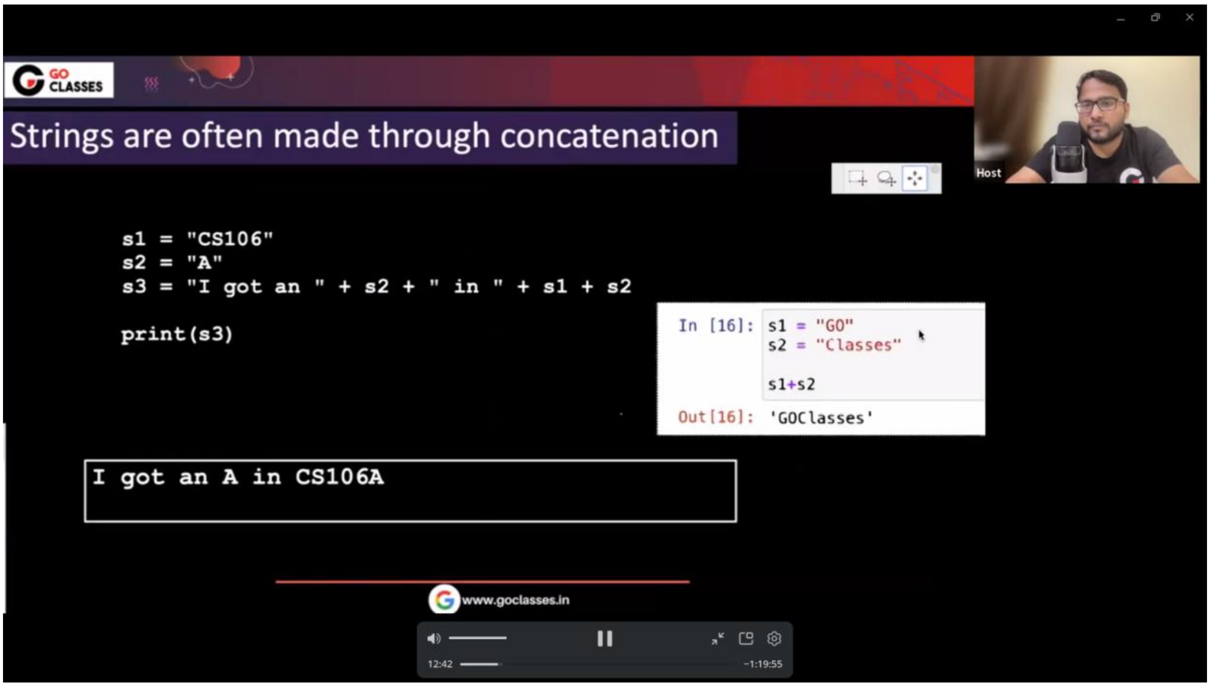
www.goclasses.in



Strings are often made through concatenation

```
s1 = "CS106"
s2 = "A"
s3 = "I got an " + s2 + " in " + s1 + s2

print(s3)
```

```
I got an A in CS106A
```

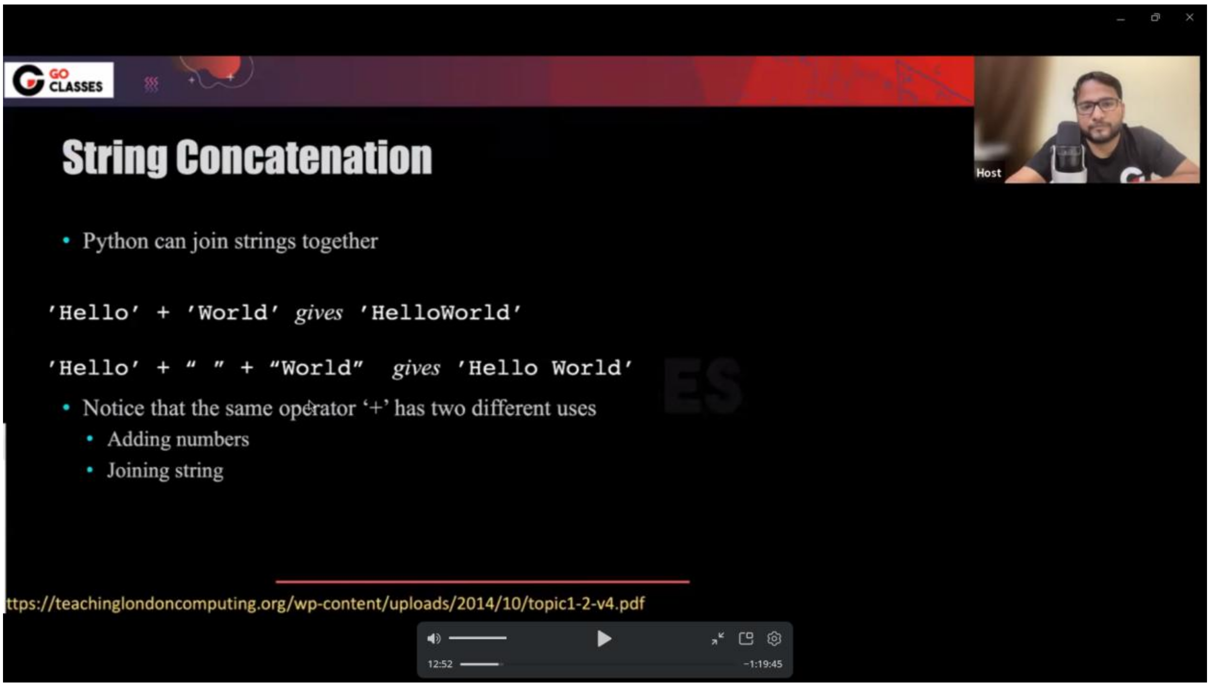


Strings are often made through concatenation

```
s1 = "CS106"
s2 = "A"
s3 = "I got an " + s2 + " in " + s1 + s2

print(s3)
```

```
In [16]: s1 = "GO"
         s2 = "Classes"

         s1+s2
Out[16]: 'GOClasses'
```

```
I got an A in CS106A
```



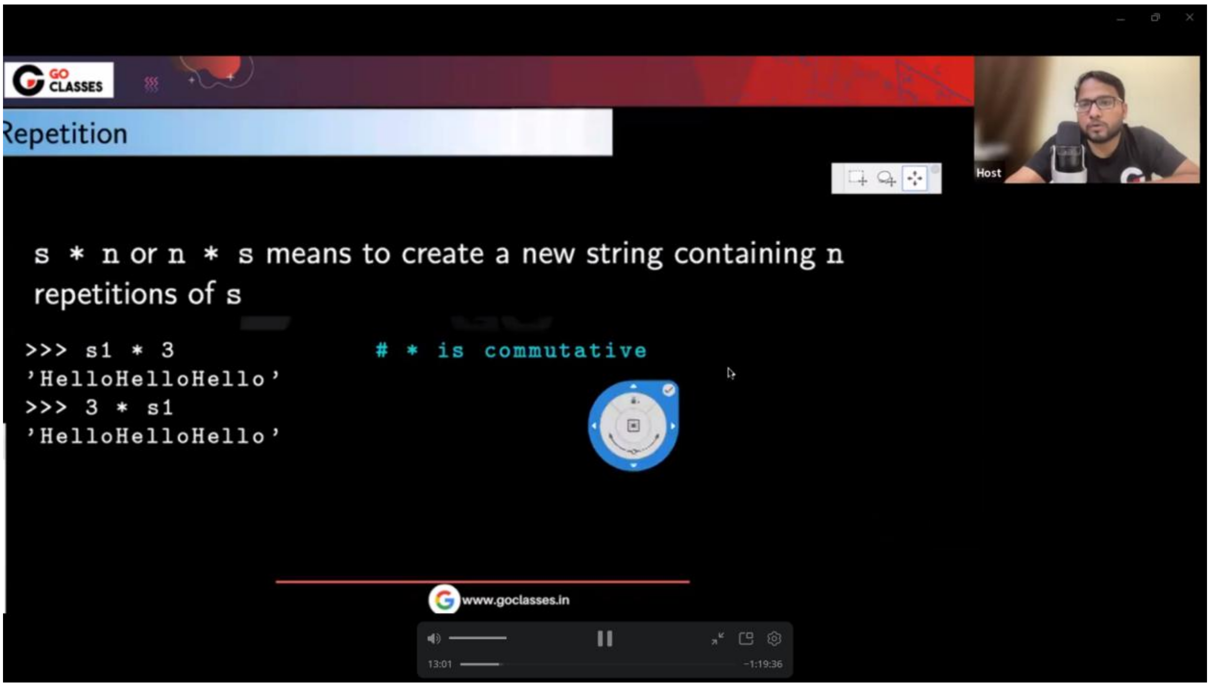
String Concatenation

- Python can join strings together

```
'Hello' + 'World' gives 'HelloWorld'

'Hello' + " " + "World"  gives 'Hello World'
```

- Notice that the same operator '+' has two different uses
  - Adding numbers
  - Joining string

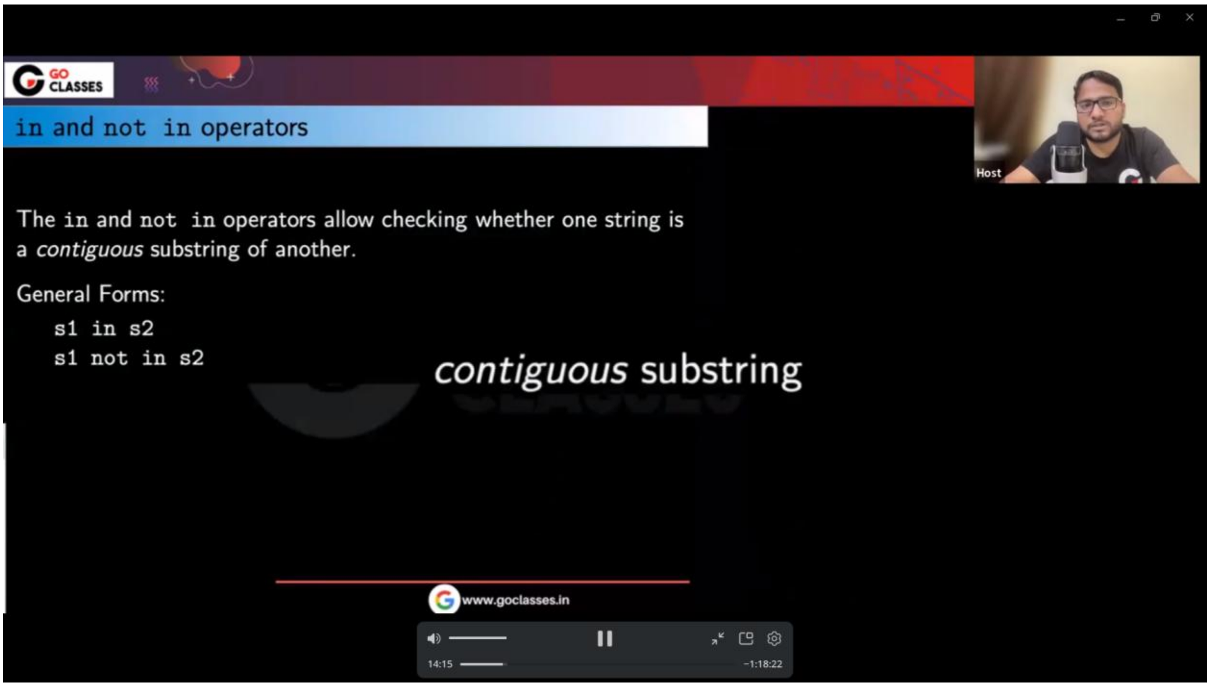
ttps://teachinglondoncomputing.org/wp-content/uploads/2014/10/topic1-2-v4.pdf



Repetition

s * n or n * s means to create a new string containing n repetitions of s

```
>>> s1 * 3           # * is commutative
'HelloHelloHello'
>>> 3 * s1
'HelloHelloHello'
```



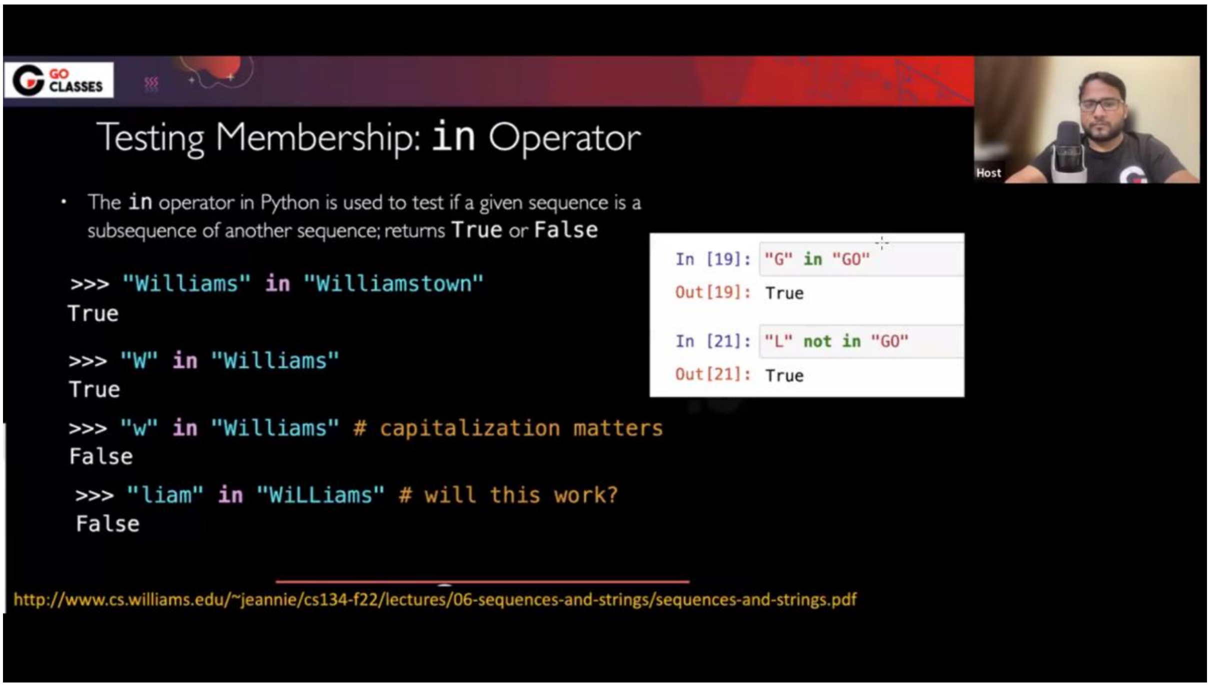
in and not in operators

The in and not in operators allow checking whether one string is a *contiguous* substring of another.

General Forms:

```
s1 in s2
s1 not in s2
```

contiguous substring



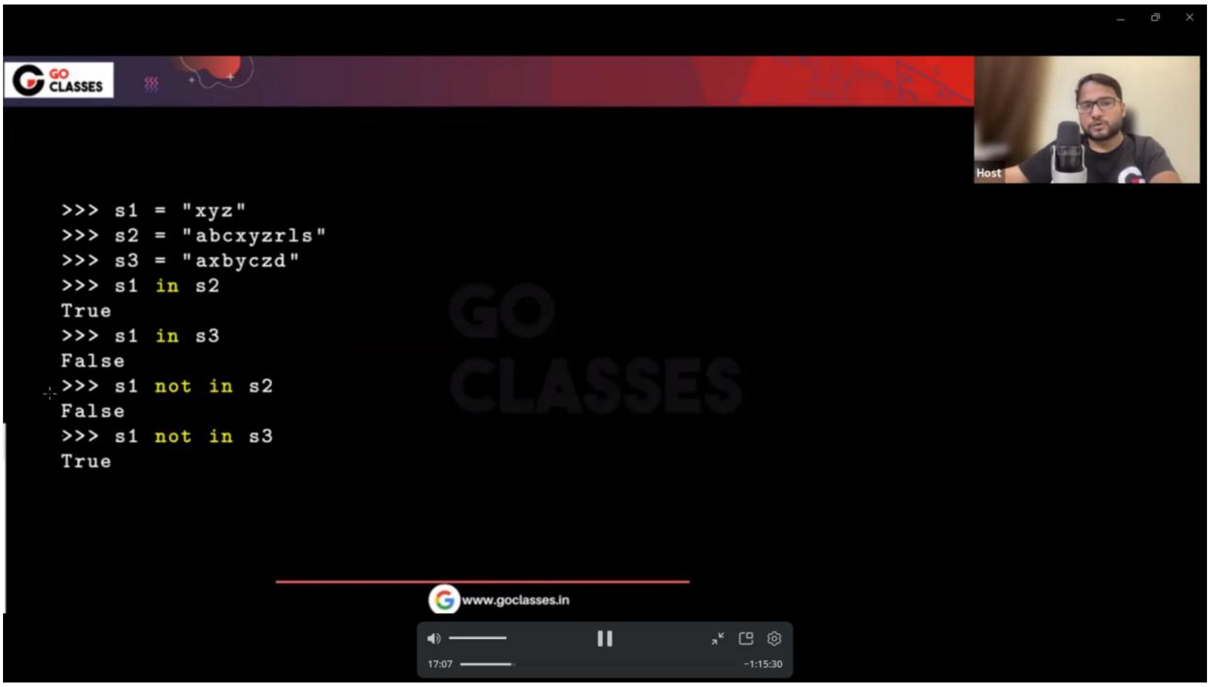
Testing Membership: in Operator

- The in operator in Python is used to test if a given sequence is a subsequence of another sequence; returns True or False

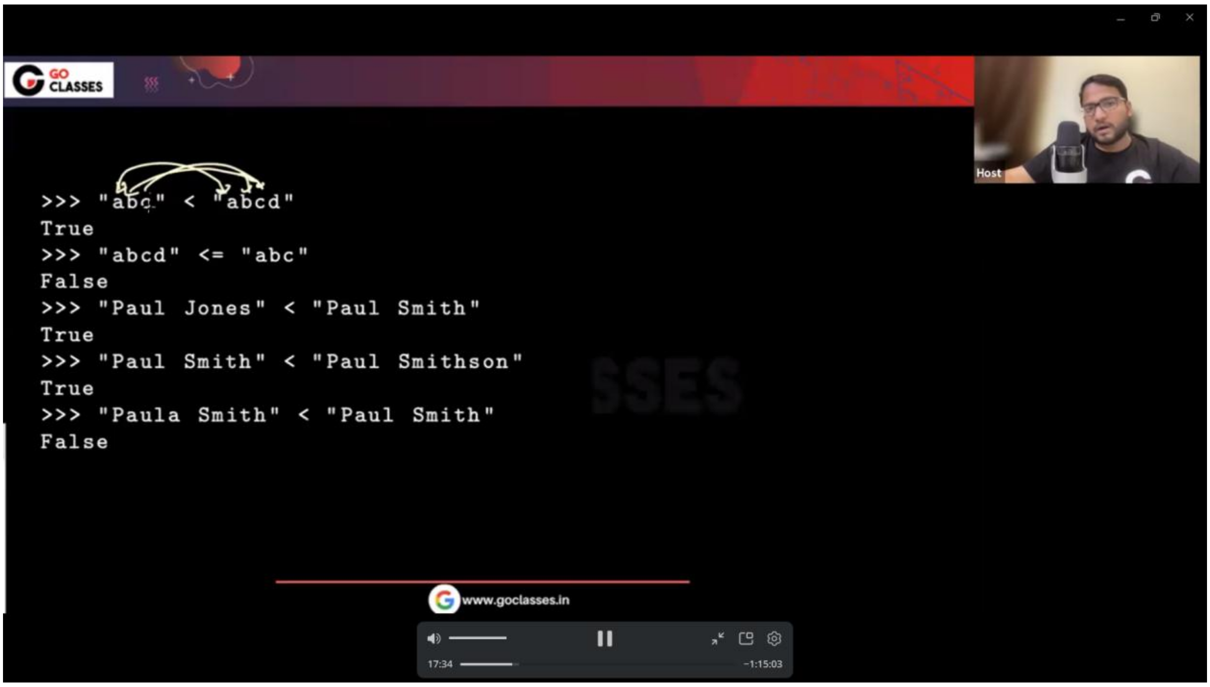
```
>>> "Williams" in "Williamstown"
True

>>> "W" in "Williams"
True

>>> "w" in "Williams" # capitalization matters
False

>>> "liam" in "WiLLiams" # will this work?
False
```

```
In [19]: "G" in "GO"
Out[19]: True

In [21]: "L" not in "GO"
Out[21]: True
```

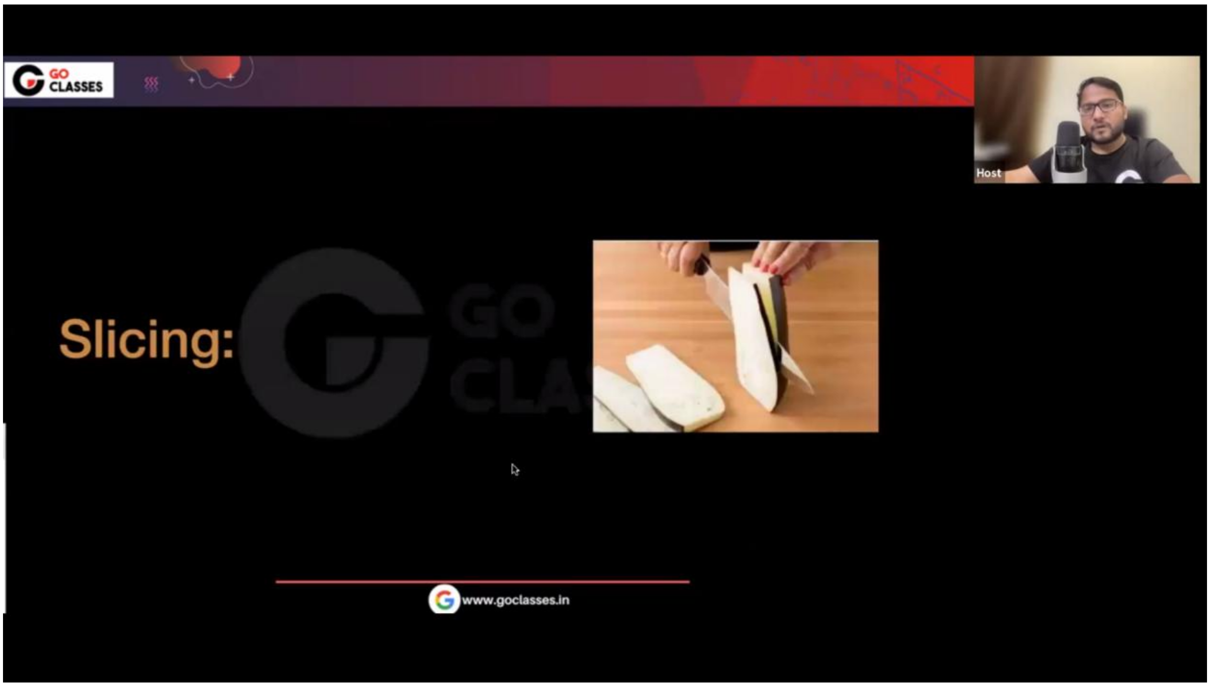
http://www.cs.williams.edu/~jeannie/cs134-f22/lectures/06-sequences-and-strings/sequences-and-strings.pdf



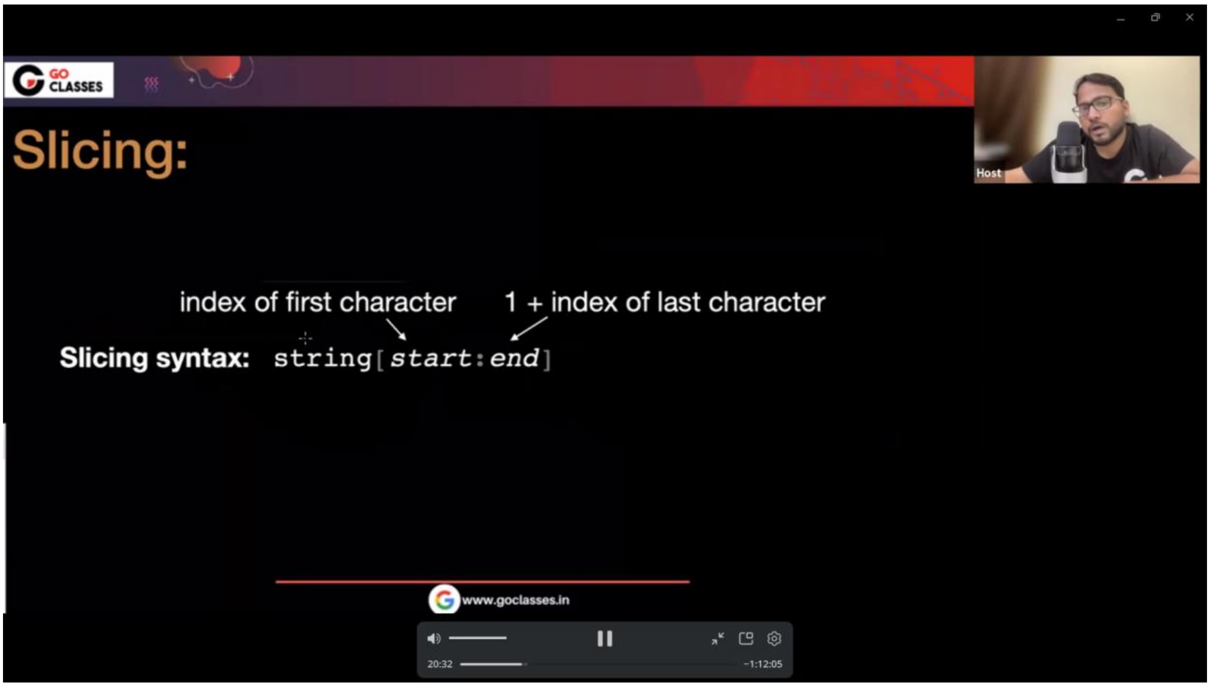
```
>>> s1 = "xyz"
>>> s2 = "abcxyzrls"
>>> s3 = "axbyczd"
>>> s1 in s2
True
>>> s1 in s3
False
>>> s1 not in s2
False
>>> s1 not in s3
True
```



```
>>> "abc" < "abcd"
True
>>> "abcd" <= "abc"
False
>>> "Paul Jones" < "Paul Smith"
True
>>> "Paul Smith" < "Paul Smithson"
True
>>> "Paula Smith" < "Paul Smith"
False
```



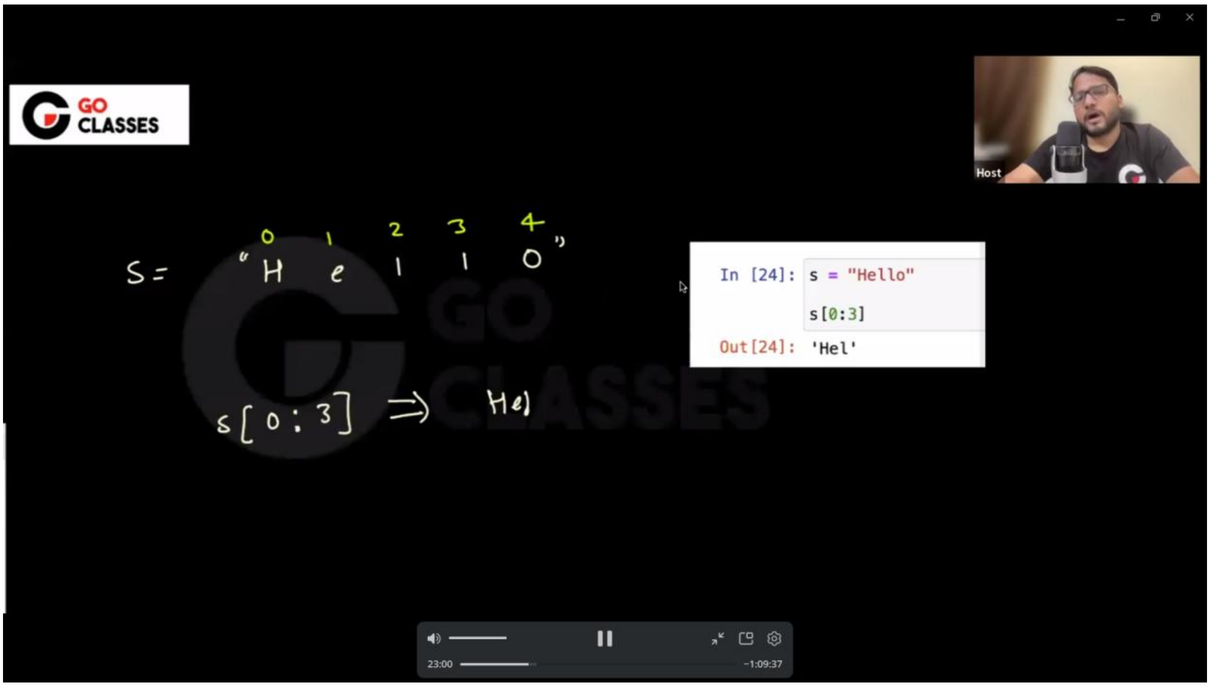
Slicing:



Slicing:

index of first character → start; 1 + index of last character → end

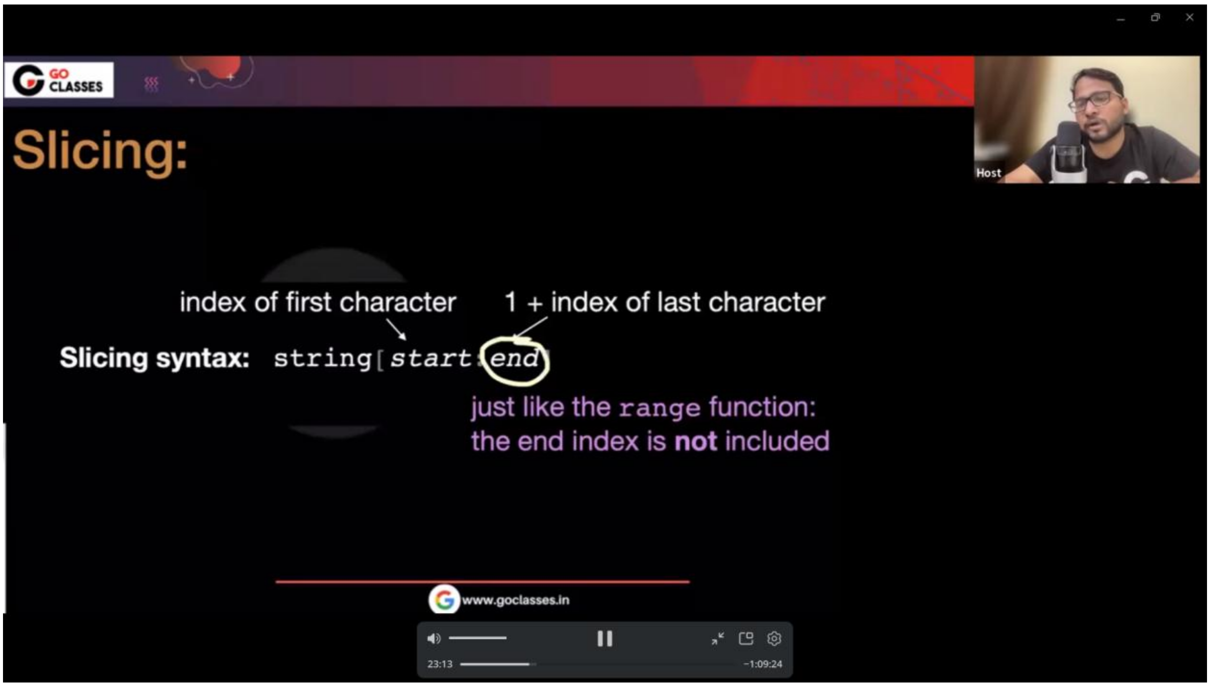
Slicing syntax: `string[start:end]`



0 1 2 3 4
s = "H e l l o"

s[0:3] ⇒ Hel

```
In [24]: s = "Hello"
         s[0:3]
Out[24]: 'Hel'
```



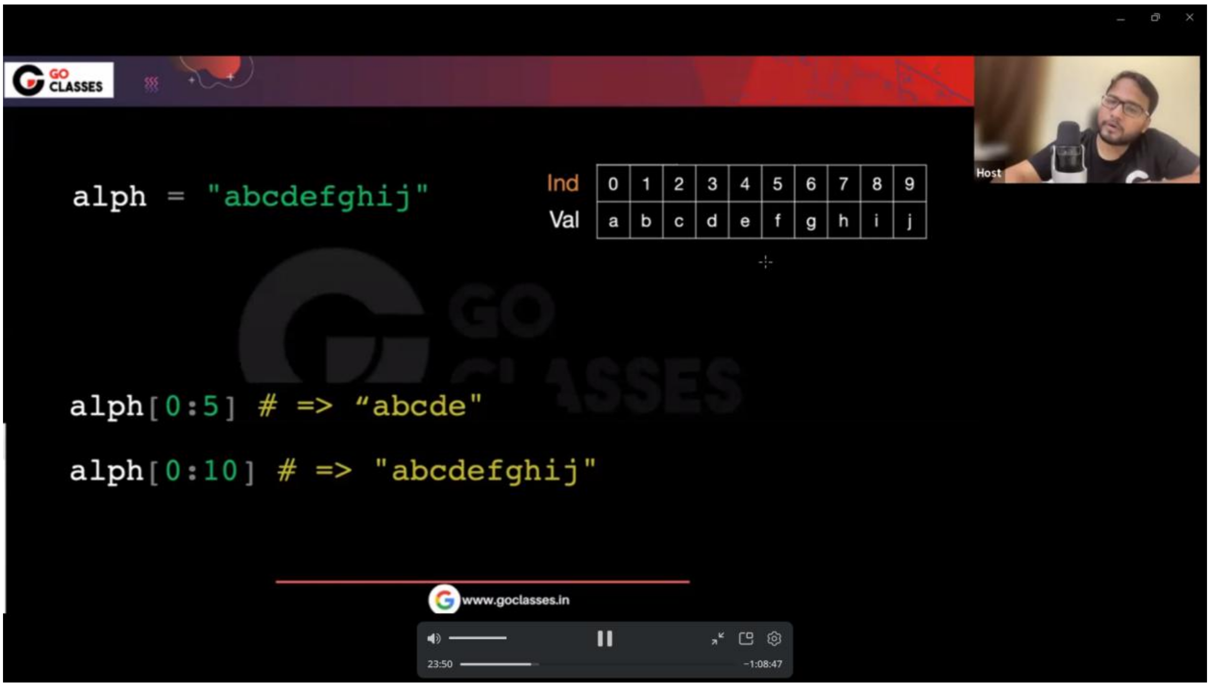
Slicing:

index of first character

1 + index of last character

Slicing syntax: `string[start:end]`

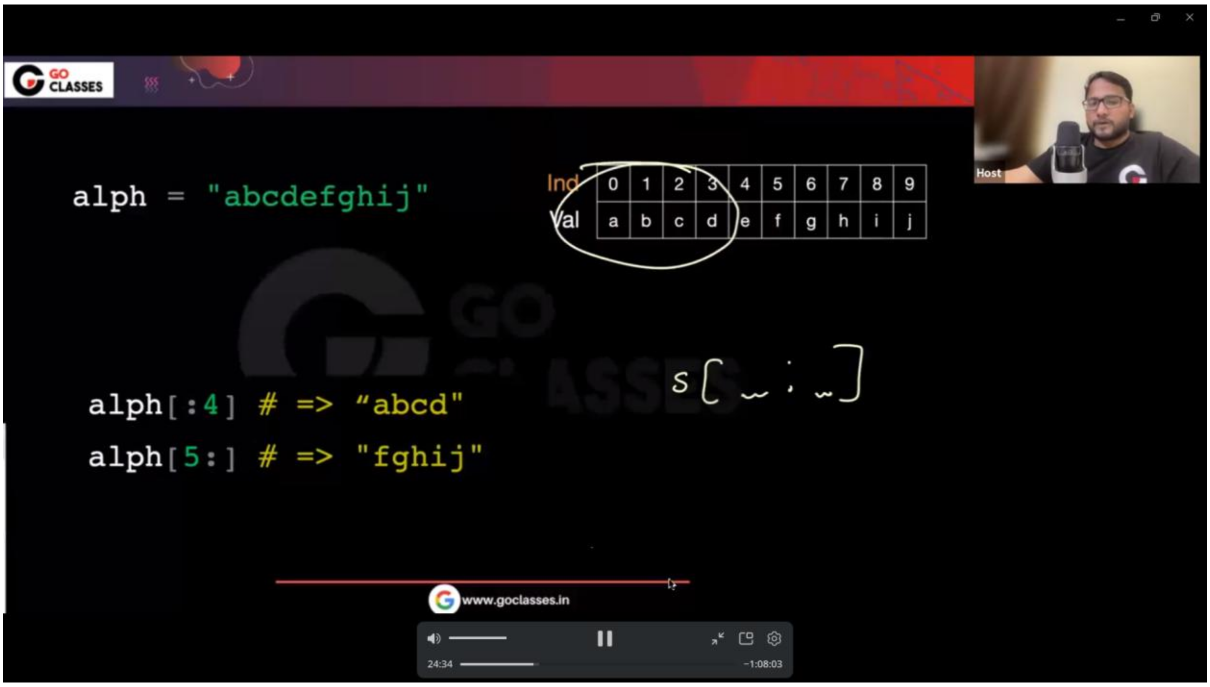
just like the `range` function: the end index is **not** included



```
alph = "abcdefghij"
```

Ind	0	1	2	3	4	5	6	7	8	9
Val	a	b	c	d	e	f	g	h	i	j

```
alph[0:5] # => "abcde"
alph[0:10] # => "abcdefghij"
```

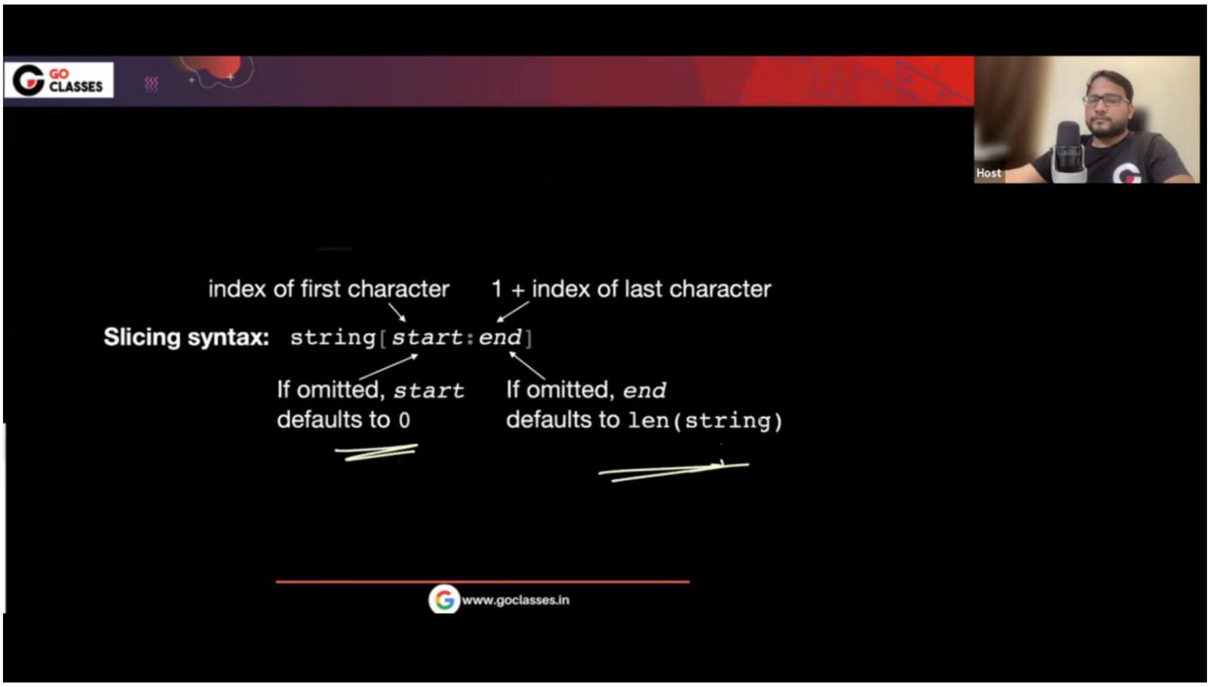


```
alph = "abcdefghij"
```

Ind	0	1	2	3	4	5	6	7	8	9
Val	a	b	c	d	e	f	g	h	i	j

s[_ : _]

```
alph[:4] # => "abcd"
alph[5:] # => "fghij"
```



index of first character

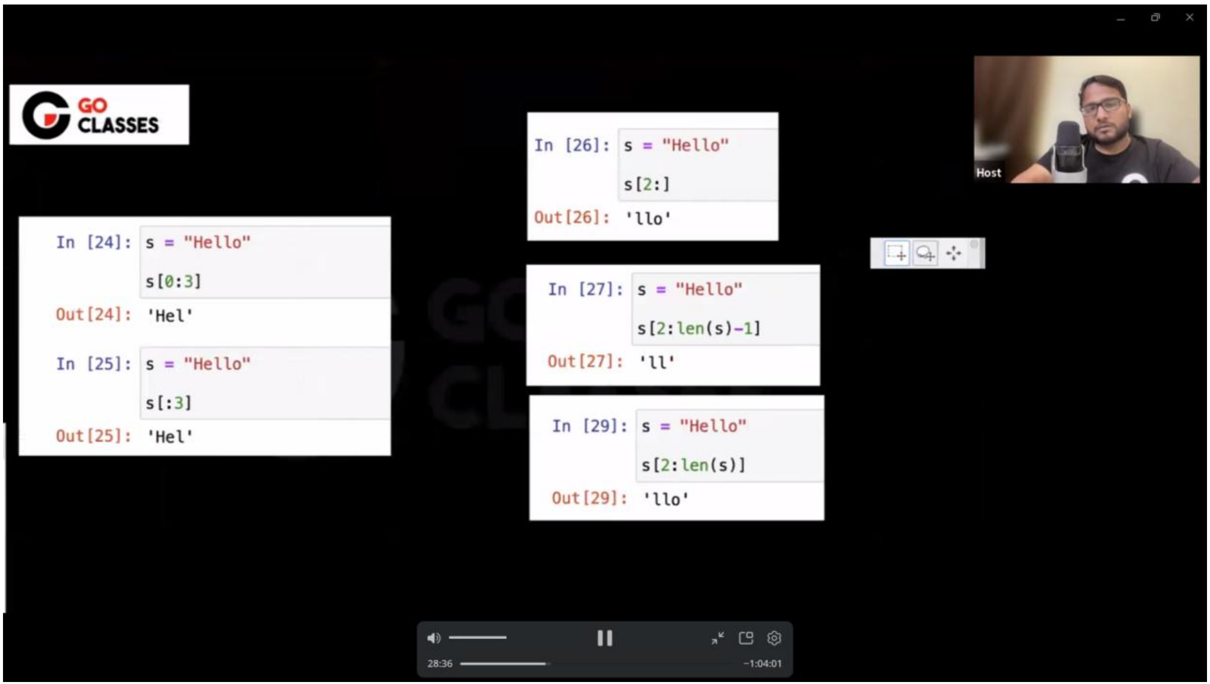
1 + index of last character

Slicing syntax: `string[start:end]`

If omitted, *start* defaults to `0`

If omitted, *end* defaults to `len(string)`

www.goclasses.in



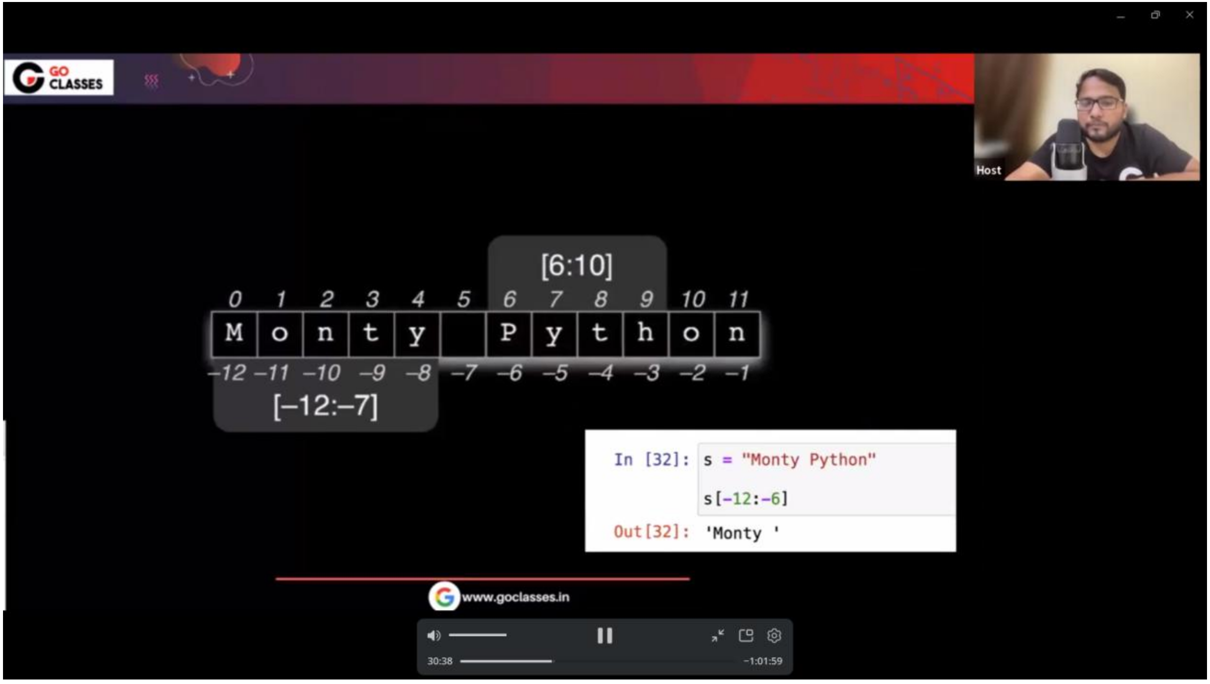
```
In [24]: s = "Hello"
         s[0:3]
Out[24]: 'Hel'
```

```
In [25]: s = "Hello"
         s[:3]
Out[25]: 'Hel'
```

```
In [26]: s = "Hello"
         s[2:]
Out[26]: 'llo'
```

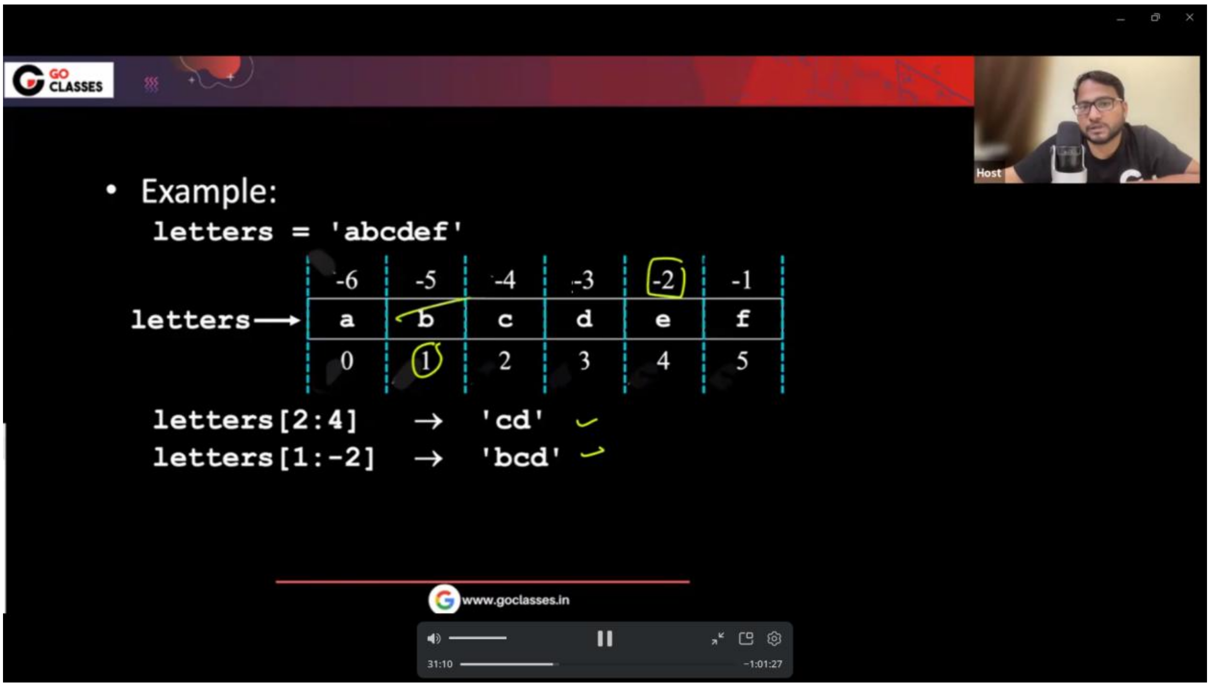
```
In [27]: s = "Hello"
         s[2:len(s)-1]
Out[27]: 'll'
```

```
In [29]: s = "Hello"
         s[2:len(s)]
Out[29]: 'llo'
```



```
                        [6:10]
  0   1   2   3   4   5   6   7   8   9  10  11
  M   o   n   t   y       P   y   t   h   o   n
-12 -11 -10  -9  -8  -7  -6  -5  -4  -3  -2  -1
      [-12:-7]
```

```
In [32]: s = "Monty Python"
         s[-12:-6]
Out[32]: 'Monty '
```

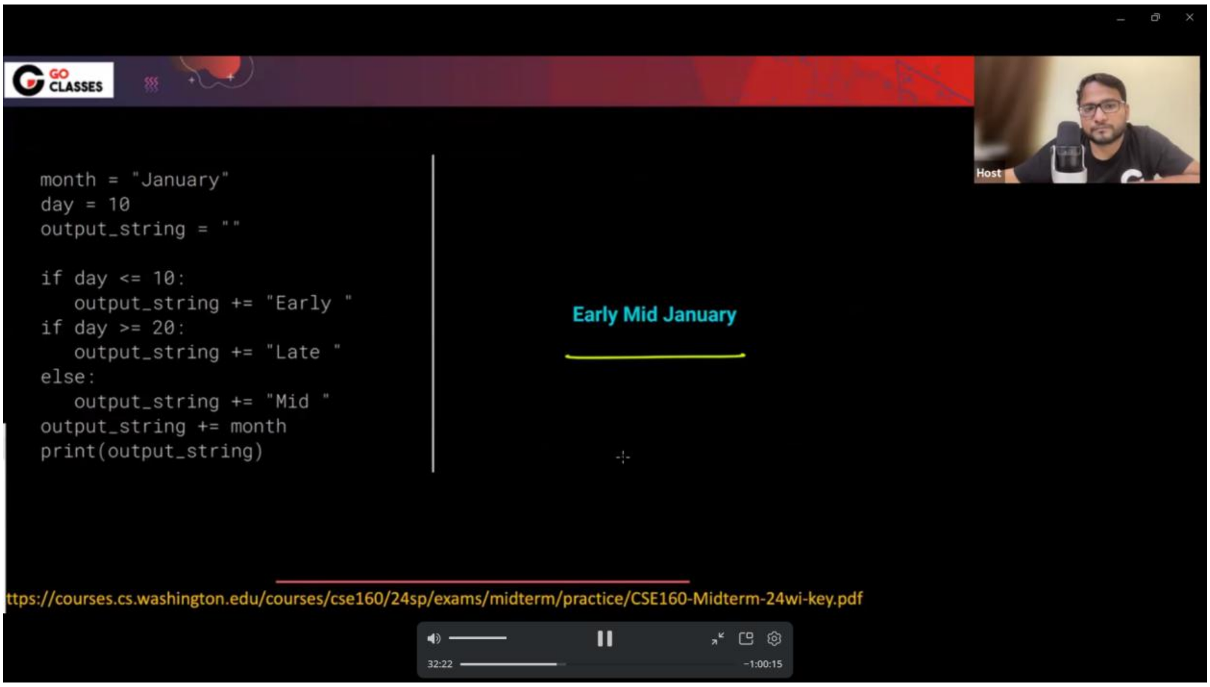


- Example:

```
letters = 'abcdef'
```

	-6	-5	-4	-3	-2	-1
letters ⟶	a	b	c	d	e	f
	0	1	2	3	4	5

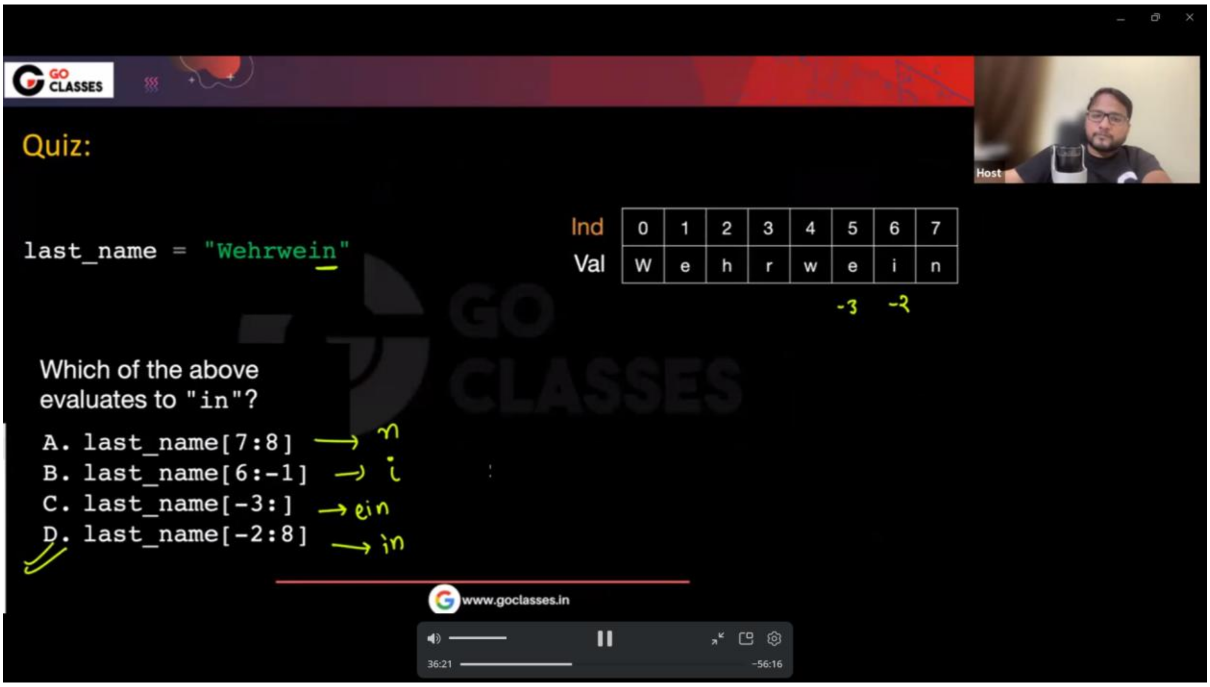
```
letters[2:4]   →  'cd'
letters[1:-2]  →  'bcd'
```



```
month = "January"
day = 10
output_string = ""

if day <= 10:
   output_string += "Early "
if day >= 20:
   output_string += "Late "
else:
   output_string += "Mid "
output_string += month
print(output_string)
```

Early Mid January

ttps://courses.cs.washington.edu/courses/cse160/24sp/exams/midterm/practice/CSE160-Midterm-24wi-key.pdf



Quiz:

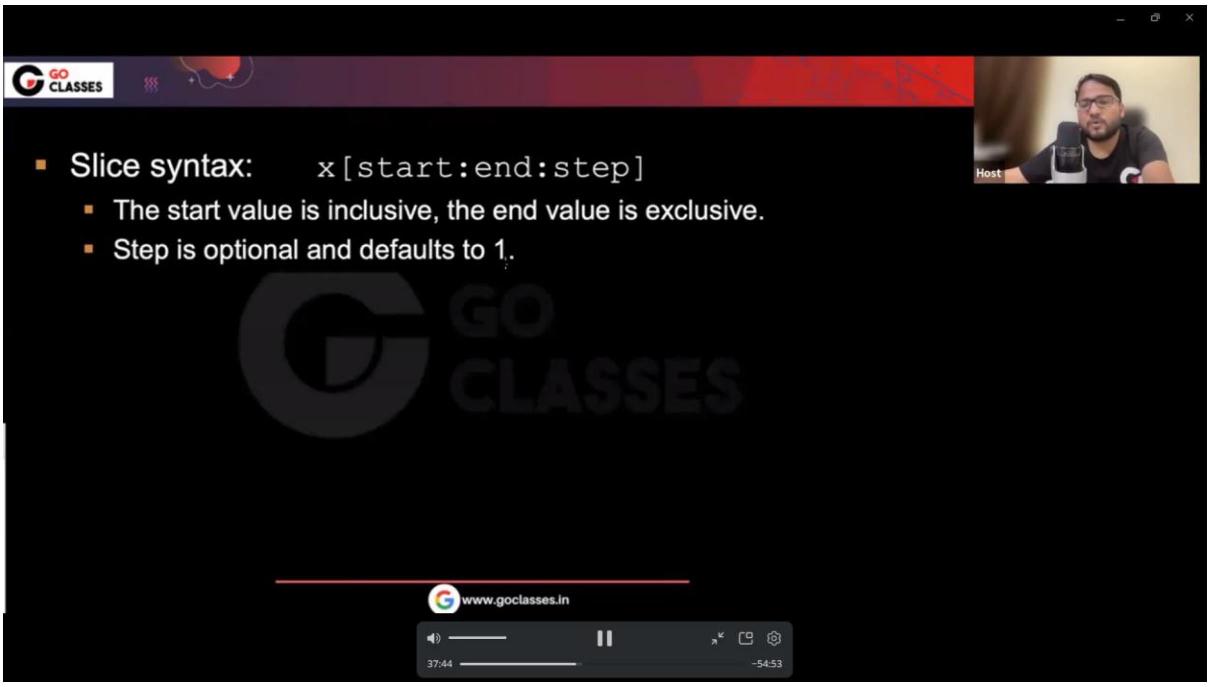
```
last_name = "Wehrwein"
```

Ind	0	1	2	3	4	5	6	7
Val	W	e	h	r	w	e	i	n
						-3	-2	

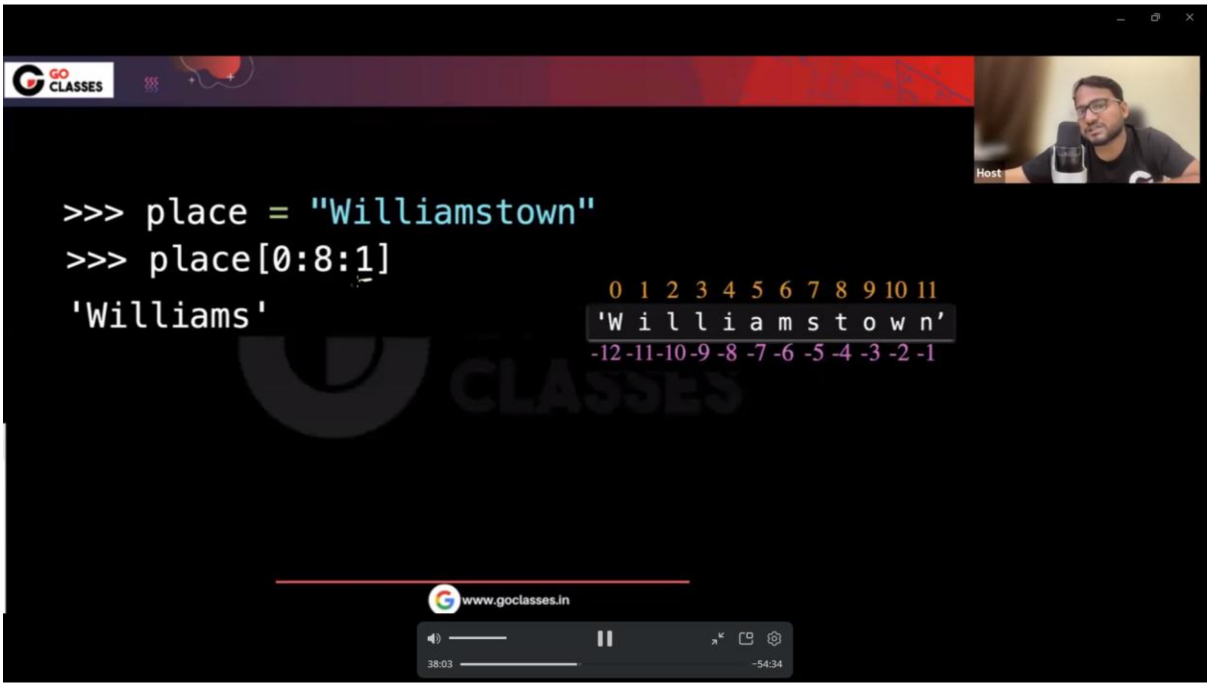
Which of the above evaluates to "in"?

```
A. last_name[7:8]  → n
B. last_name[6:-1] → i
C. last_name[-3:]  → ein
D. last_name[-2:8] → in
```

In Python, the slice `s[3:1]` will result in an empty sequence (e.g., an empty string for a string, an empty list for a list), not an error.

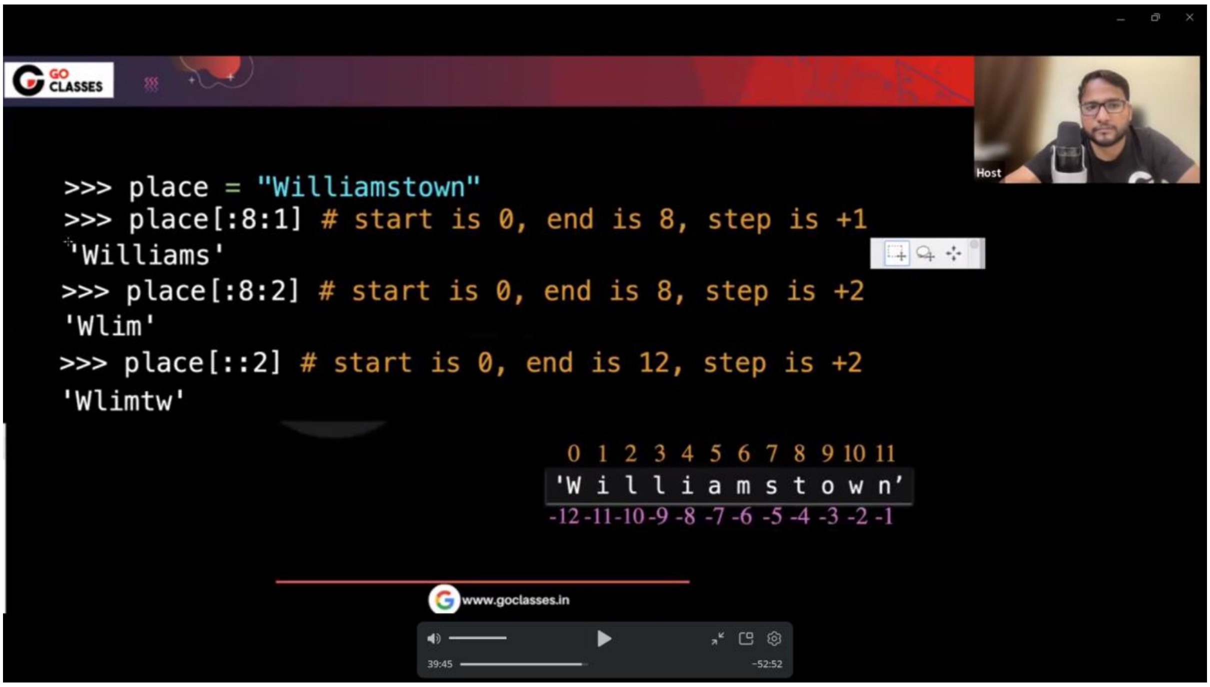


- Slice syntax: `x[start:end:step]`
  - The start value is inclusive, the end value is exclusive.
  - Step is optional and defaults to 1.



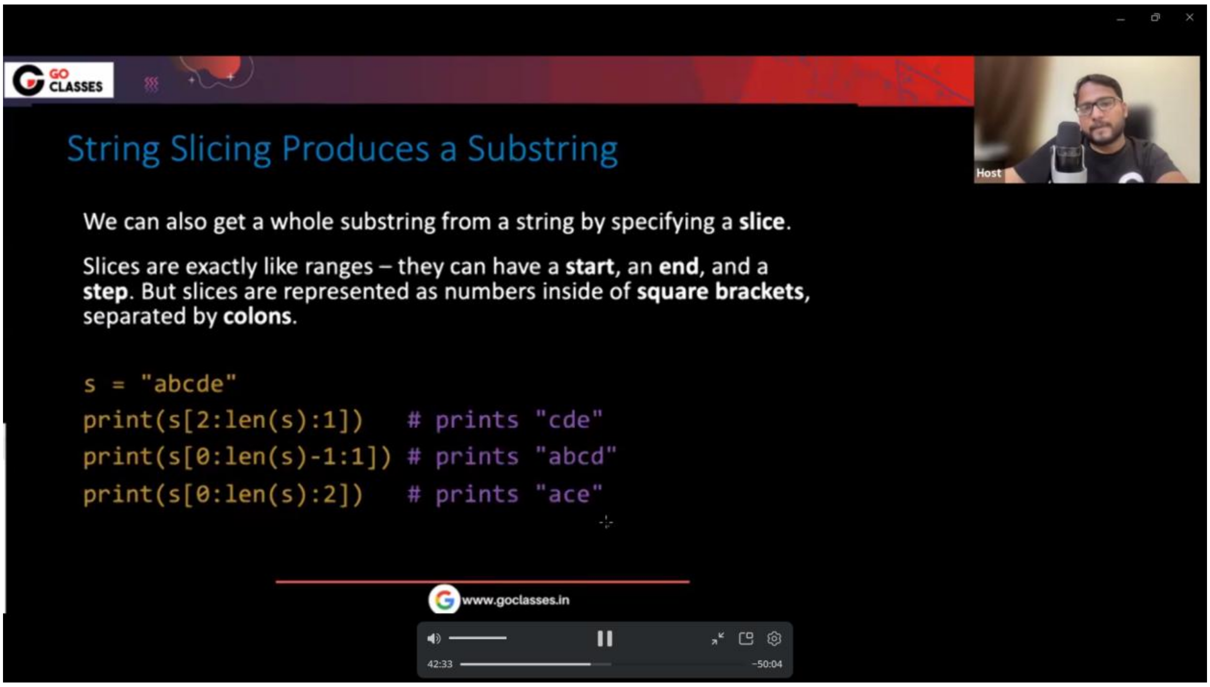
```
>>> place = "Williamstown"
>>> place[0:8:1]
'Williams'
```

```
 0  1  2  3  4  5  6  7  8  9 10 11
'W  i  l  l  i  a  m  s  t  o  w  n'
-12-11-10-9 -8 -7 -6 -5 -4 -3 -2 -1
```



```
>>> place = "Williamstown"
>>> place[:8:1] # start is 0, end is 8, step is +1
'Williams'
>>> place[:8:2] # start is 0, end is 8, step is +2
'Wlim'
>>> place[::2] # start is 0, end is 12, step is +2
'Wlimtw'
```

```
 0  1  2  3  4  5  6  7  8  9 10 11
'W  i  l  l  i  a  m  s  t  o  w  n'
-12-11-10-9 -8 -7 -6 -5 -4 -3 -2 -1
```

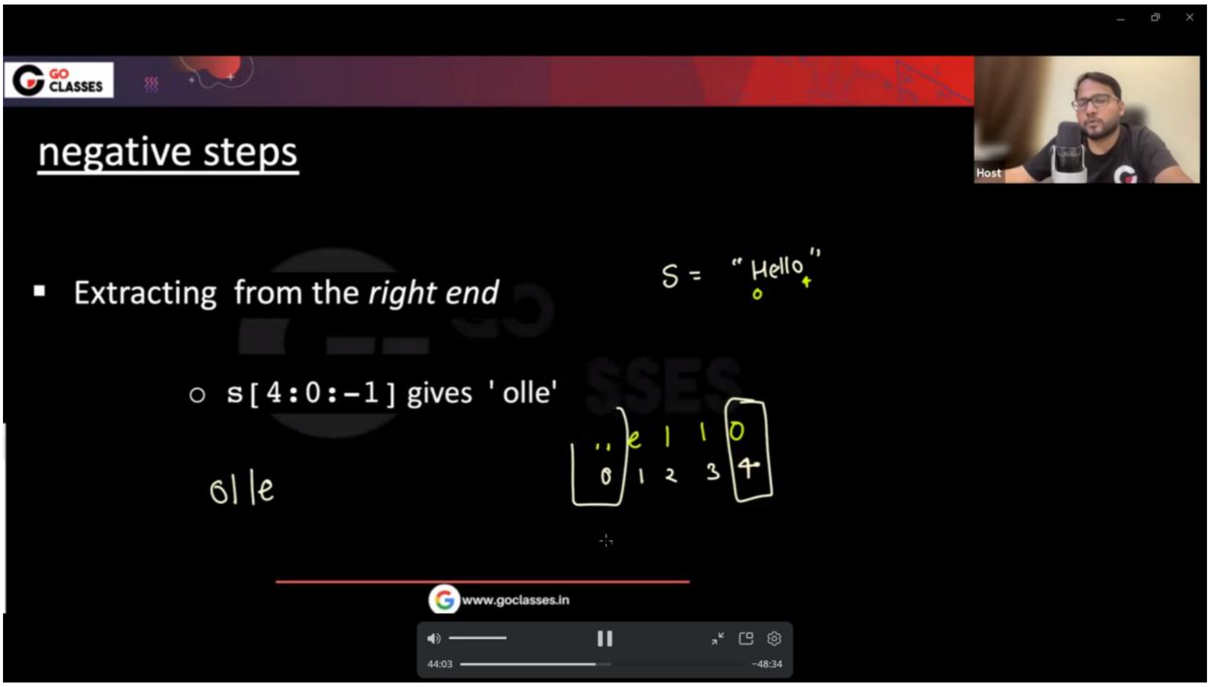


String Slicing Produces a Substring

We can also get a whole substring from a string by specifying a **slice**.

Slices are exactly like ranges – they can have a **start**, an **end**, and a **step**. But slices are represented as numbers inside of **square brackets**, separated by **colons**.

```
s = "abcde"
print(s[2:len(s):1])   # prints "cde"
print(s[0:len(s)-1:1]) # prints "abcd"
print(s[0:len(s):2])   # prints "ace"
```

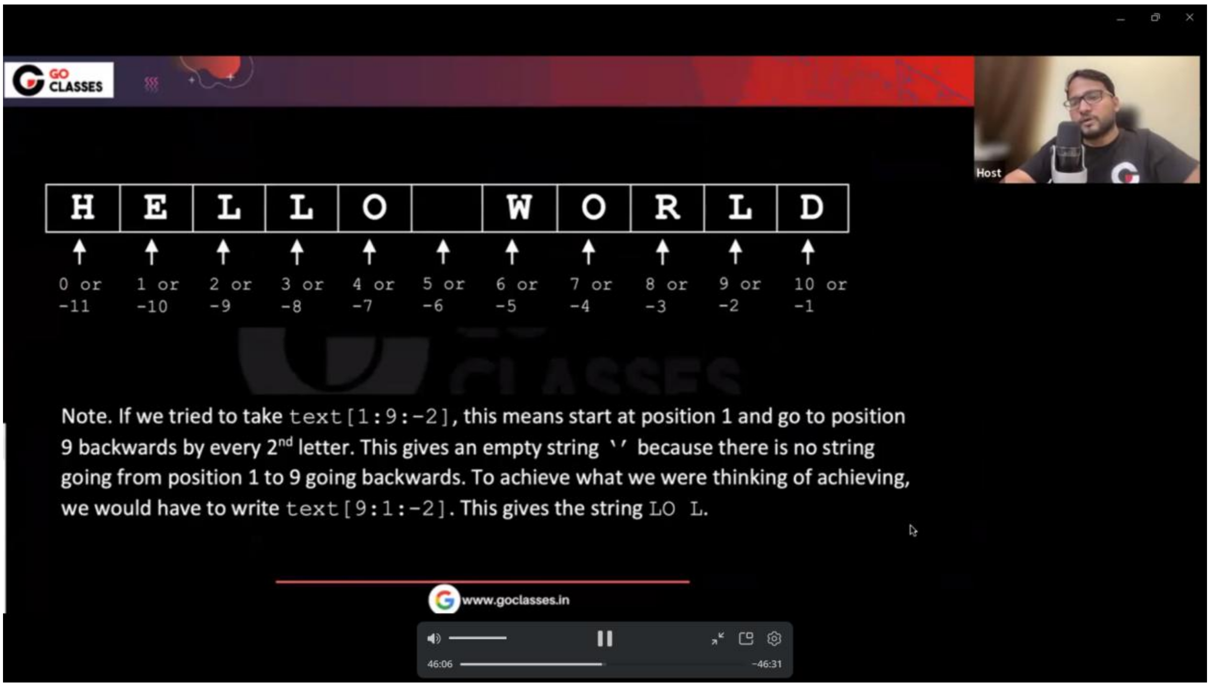


negative steps

- Extracting from the *right end*
  - s[4:0:-1] gives 'olle'

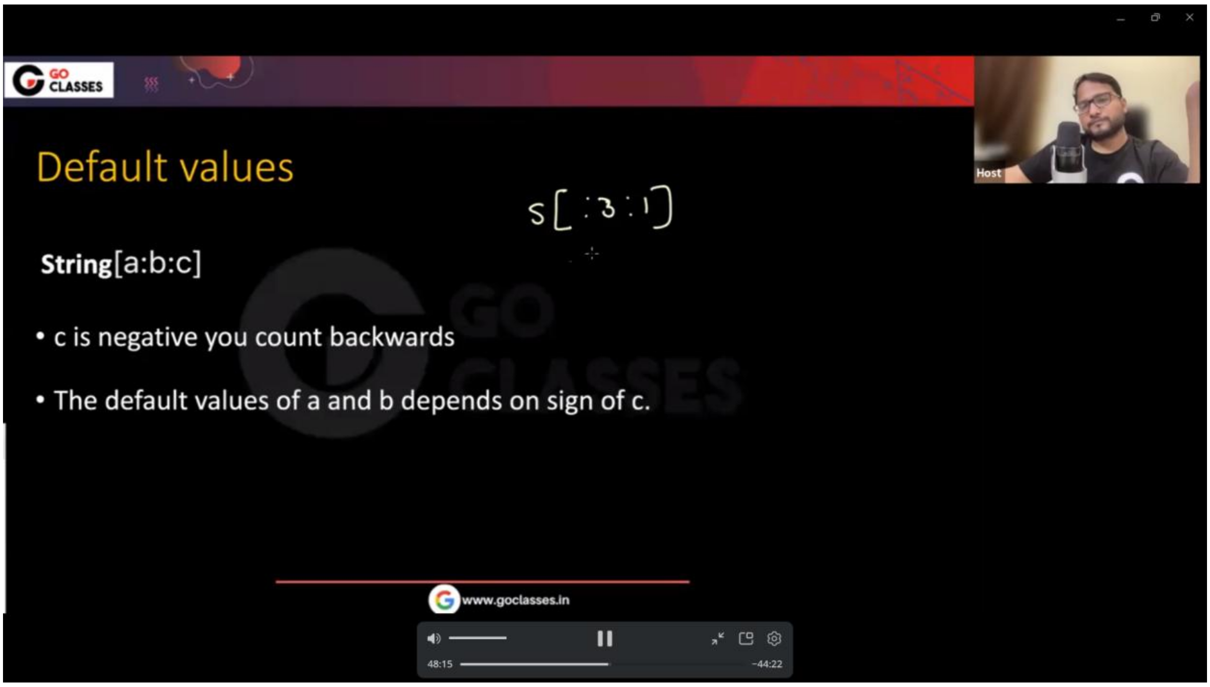
ol le

S = "Hello"



H	E	L	L	O		W	O	R	L	D
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
0 or -11	1 or -10	2 or -9	3 or -8	4 or -7	5 or -6	6 or -5	7 or -4	8 or -3	9 or -2	10 or -1

Note. If we tried to take `text[1:9:-2]`, this means start at position 1 and go to position 9 backwards by every 2nd letter. This gives an empty string `''` because there is no string going from position 1 to 9 going backwards. To achieve what we were thinking of achieving, we would have to write `text[9:1:-2]`. This gives the string `LO L`.

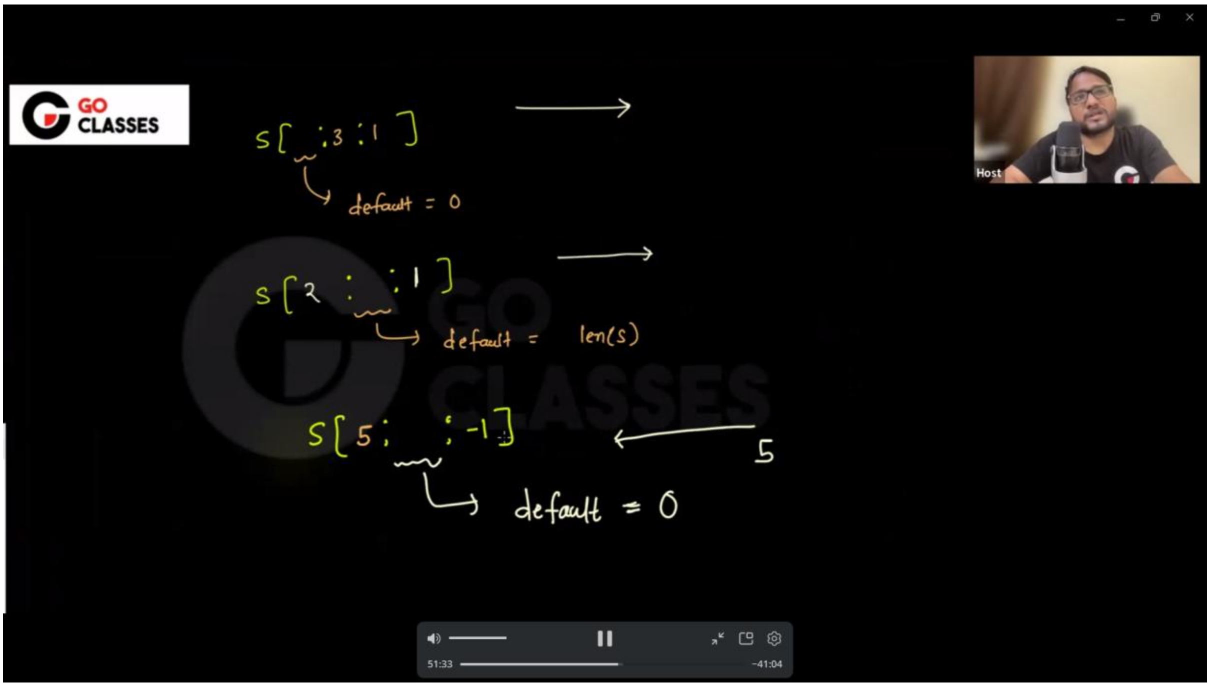


Default values

s[:3:1]

String[a:b:c]

- c is negative you count backwards
- The default values of a and b depends on sign of c.



s[:3 :1] →

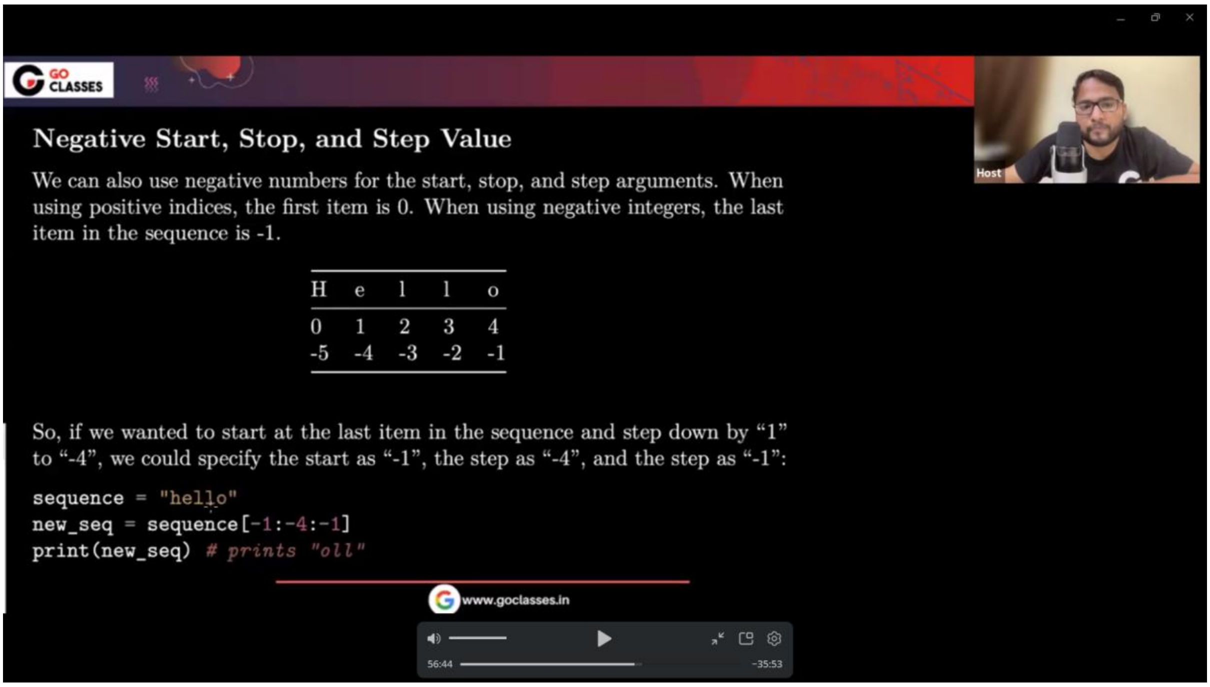
default = 0

s[2 : :1] →

default = len(s)

s[5: :-1] ← 5

default = 0



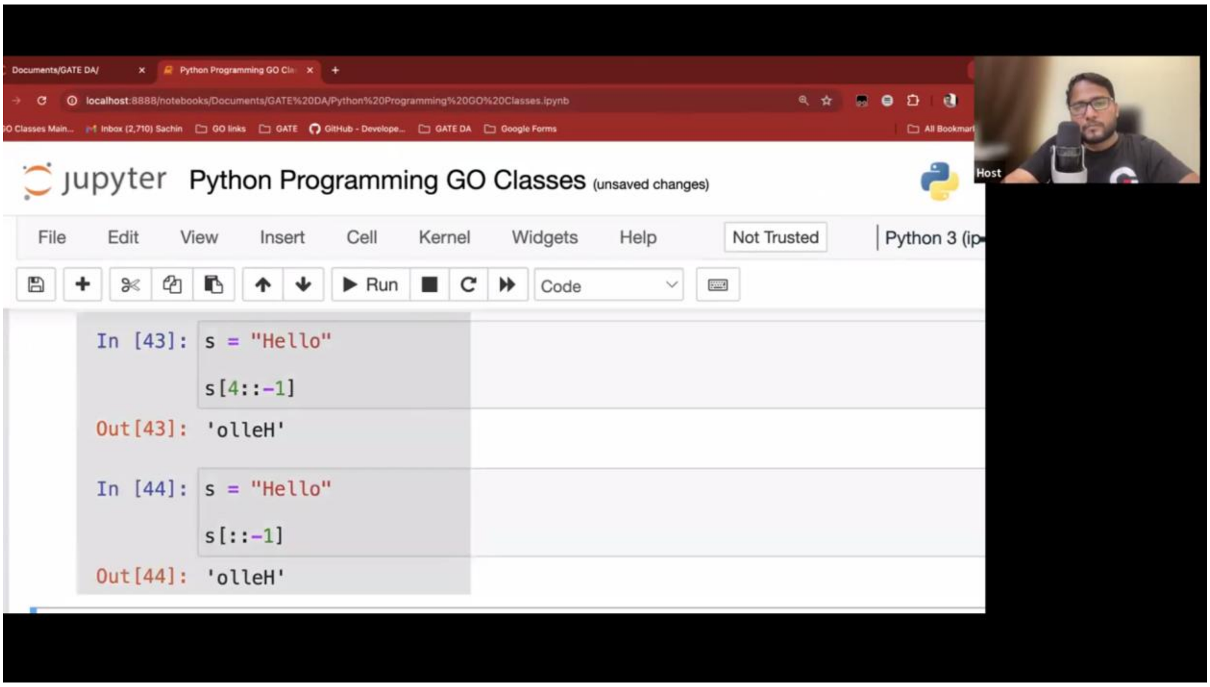
Negative Start, Stop, and Step Value

We can also use negative numbers for the start, stop, and step arguments. When using positive indices, the first item is 0. When using negative integers, the last item in the sequence is -1.

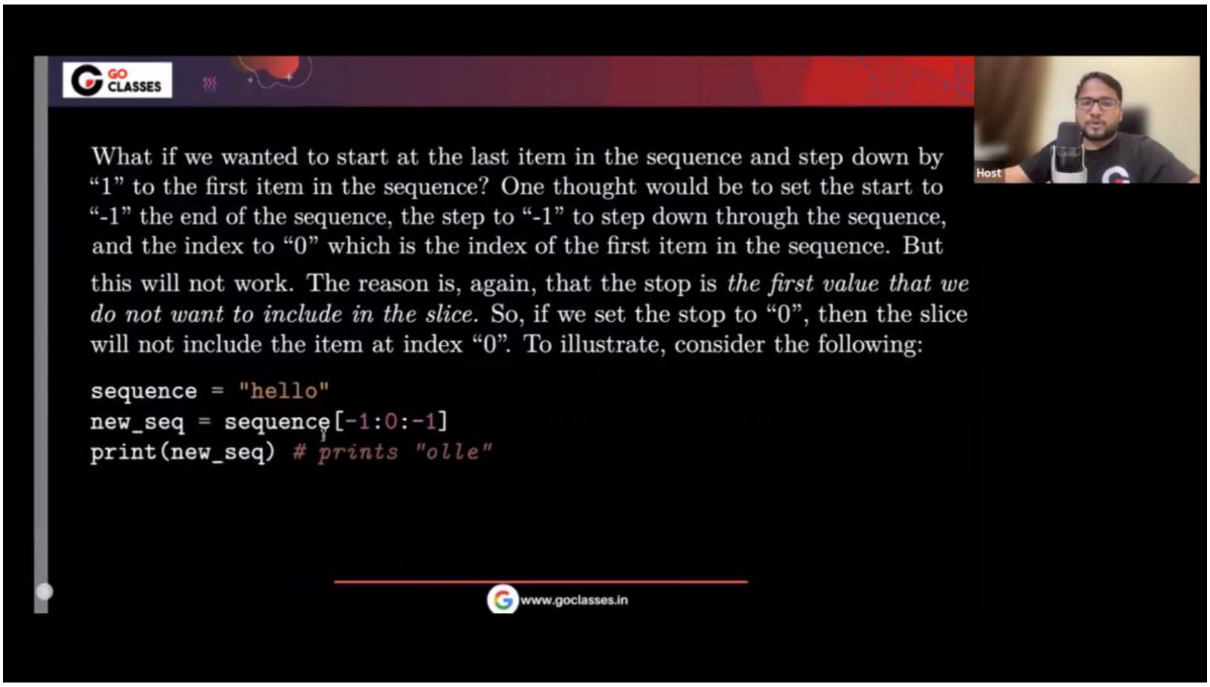
H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

So, if we wanted to start at the last item in the sequence and step down by "1" to "-4", we could specify the start as "-1", the step as "-4", and the step as "-1":

```
sequence = "hello"
new_seq = sequence[-1:-4:-1]
print(new_seq) # prints "oll"
```

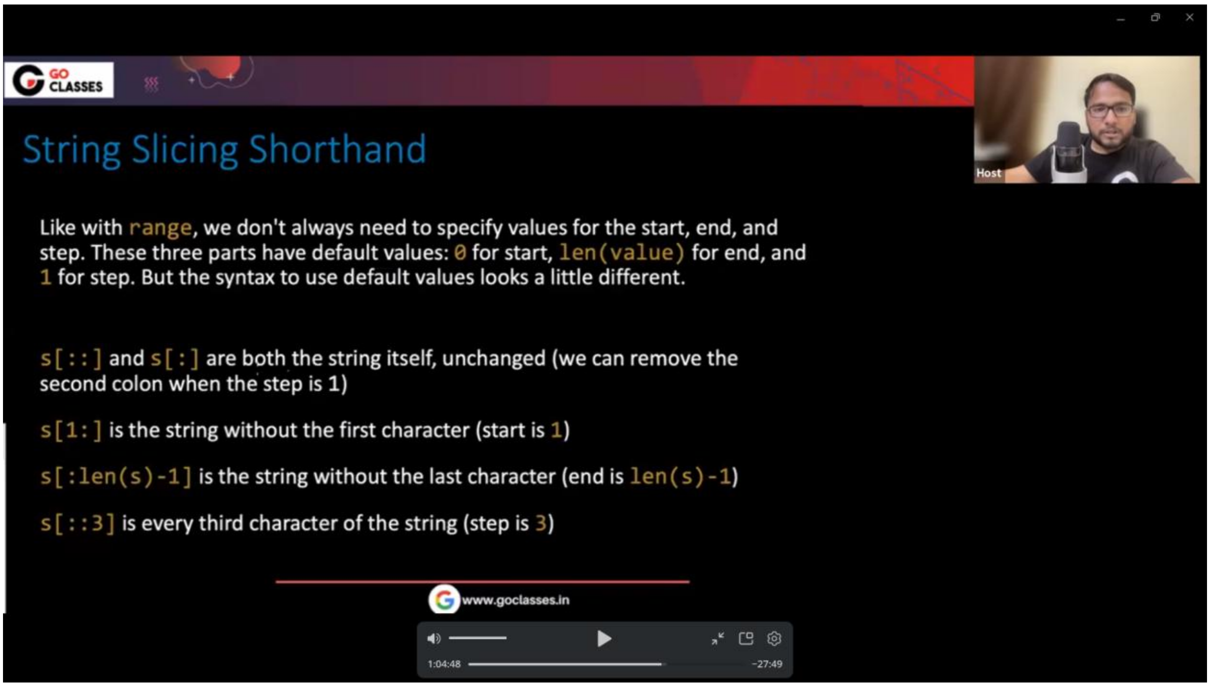


```
In [43]: s = "Hello"
         s[4::-1]
Out[43]: 'olleH'

In [44]: s = "Hello"
         s[::-1]
Out[44]: 'olleH'
```



What if we wanted to start at the last item in the sequence and step down by "1" to the first item in the sequence? One thought would be to set the start to "-1" the end of the sequence, the step to "-1" to step down through the sequence, and the index to "0" which is the index of the first item in the sequence. But this will not work. The reason is, again, that the stop is *the first value that we do not want to include in the slice.* So, if we set the stop to "0", then the slice will not include the item at index "0". To illustrate, consider the following:

```
sequence = "hello"
new_seq = sequence[-1:0:-1]
print(new_seq) # prints "olle"
```



String Slicing Shorthand

Like with range, we don't always need to specify values for the start, end, and step. These three parts have default values: 0 for start, len(value) for end, and 1 for step. But the syntax to use default values looks a little different.

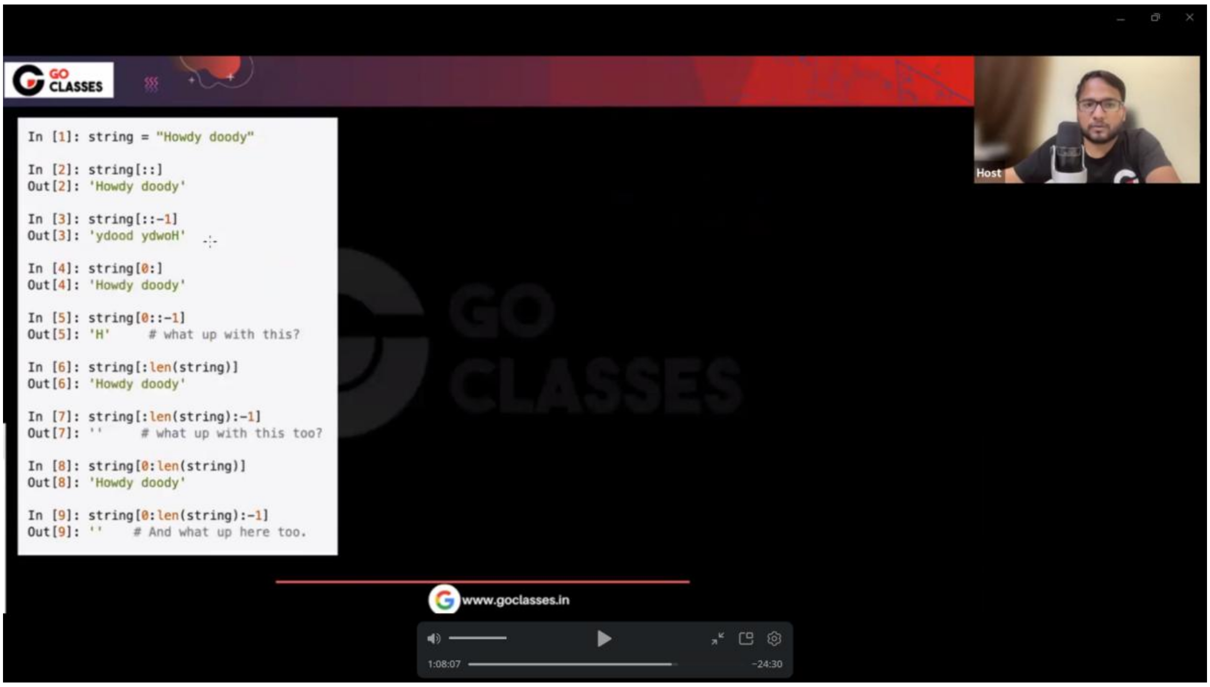
s[::] and s[:] are both the string itself, unchanged (we can remove the second colon when the step is 1)

s[1:] is the string without the first character (start is 1)

s[:len(s)-1] is the string without the last character (end is len(s)-1)

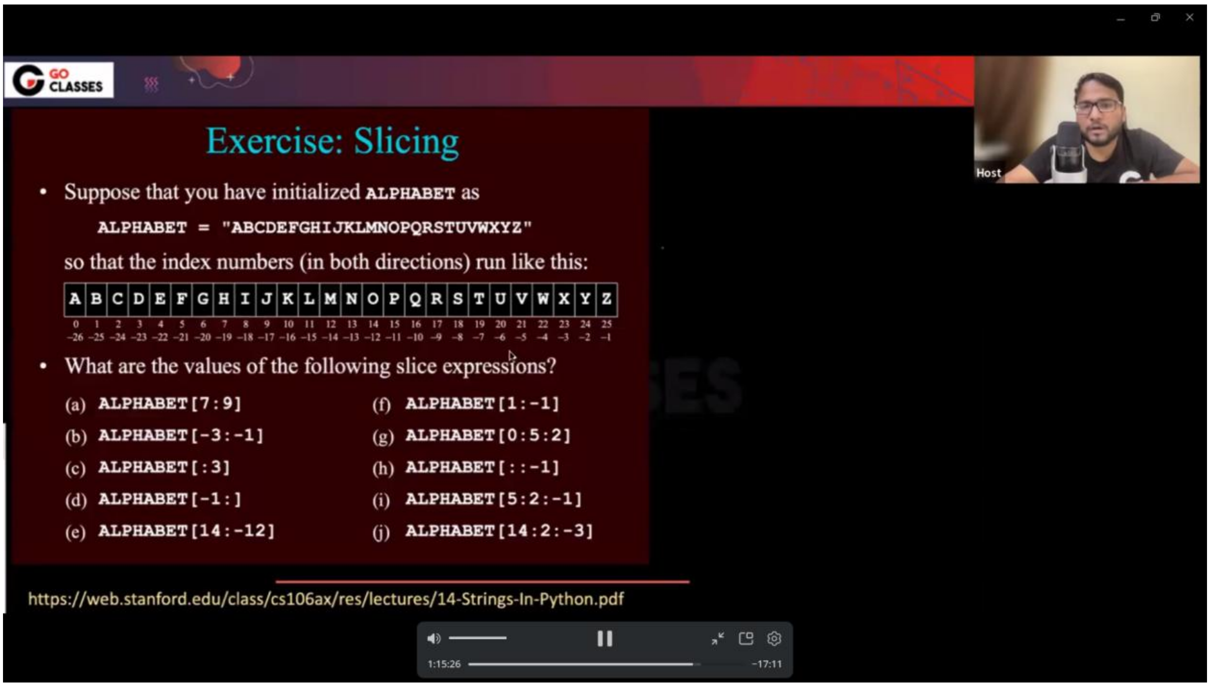
s[::3] is every third character of the string (step is 3)

www.goclasses.in



```
In [1]: string = "Howdy doody"

In [2]: string[::]
Out[2]: 'Howdy doody'

In [3]: string[::-1]
Out[3]: 'ydood ydwoH'

In [4]: string[0:]
Out[4]: 'Howdy doody'

In [5]: string[0::-1]
Out[5]: 'H'     # what up with this?

In [6]: string[:len(string)]
Out[6]: 'Howdy doody'

In [7]: string[:len(string):-1]
Out[7]: ''     # what up with this too?

In [8]: string[0:len(string)]
Out[8]: 'Howdy doody'

In [9]: string[0:len(string):-1]
Out[9]: ''    # And what up here too.
```

www.goclasses.in



Exercise: Slicing

- Suppose that you have initialized **ALPHABET** as

  ```
  ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  ```

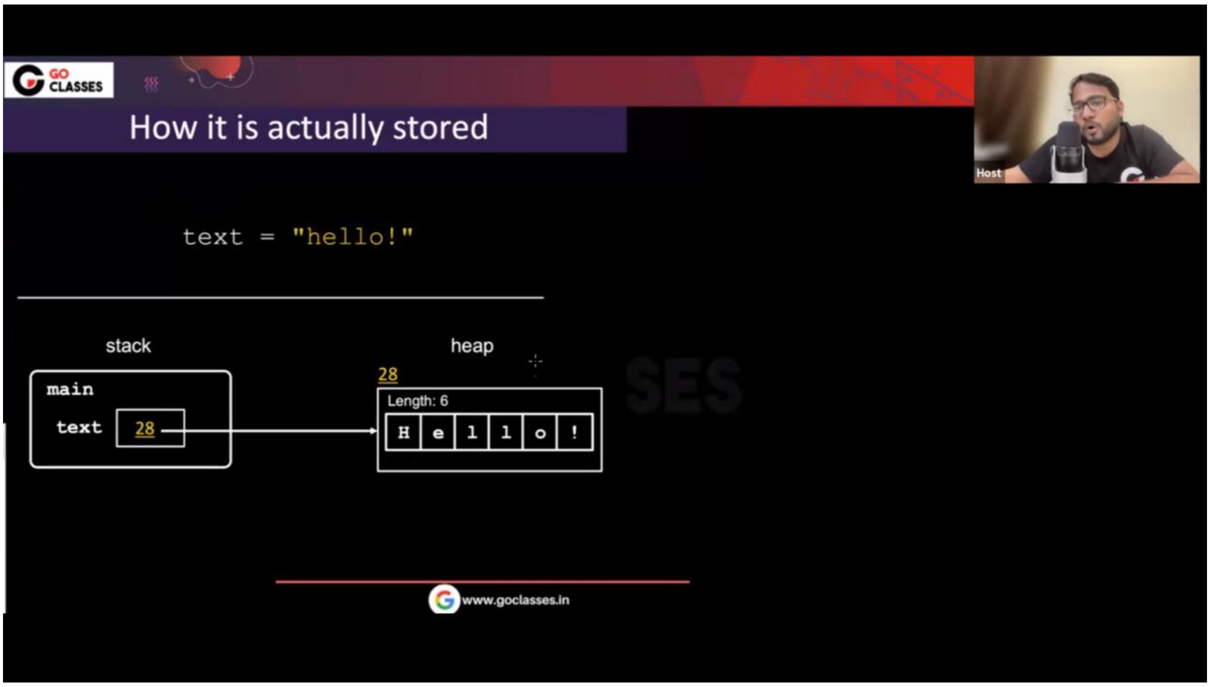
  so that the index numbers (in both directions) run like this:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
–26	–25	–24	–23	–22	–21	–20	–19	–18	–17	–16	–15	–14	–13	–12	–11	–10	–9	–8	–7	–6	–5	–4	–3	–2	–1

- What are the values of the following slice expressions?

(a) `ALPHABET[7:9]`
(b) `ALPHABET[-3:-1]`
(c) `ALPHABET[:3]`
(d) `ALPHABET[-1:]`
(e) `ALPHABET[14:-12]`
(f) `ALPHABET[1:-1]`
(g) `ALPHABET[0:5:2]`
(h) `ALPHABET[::-1]`
(i) `ALPHABET[5:2:-1]`
(j) `ALPHABET[14:2:-3]`

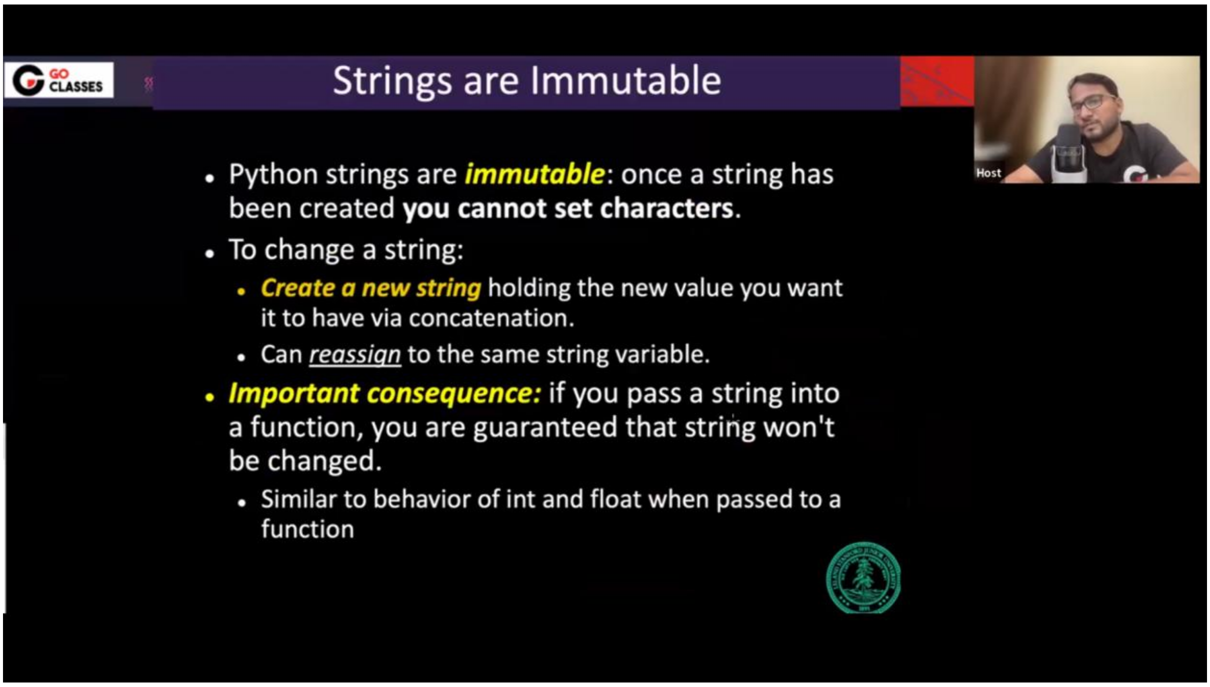
https://web.stanford.edu/class/cs106ax/res/lectures/14-Strings-In-Python.pdf



How it is actually stored

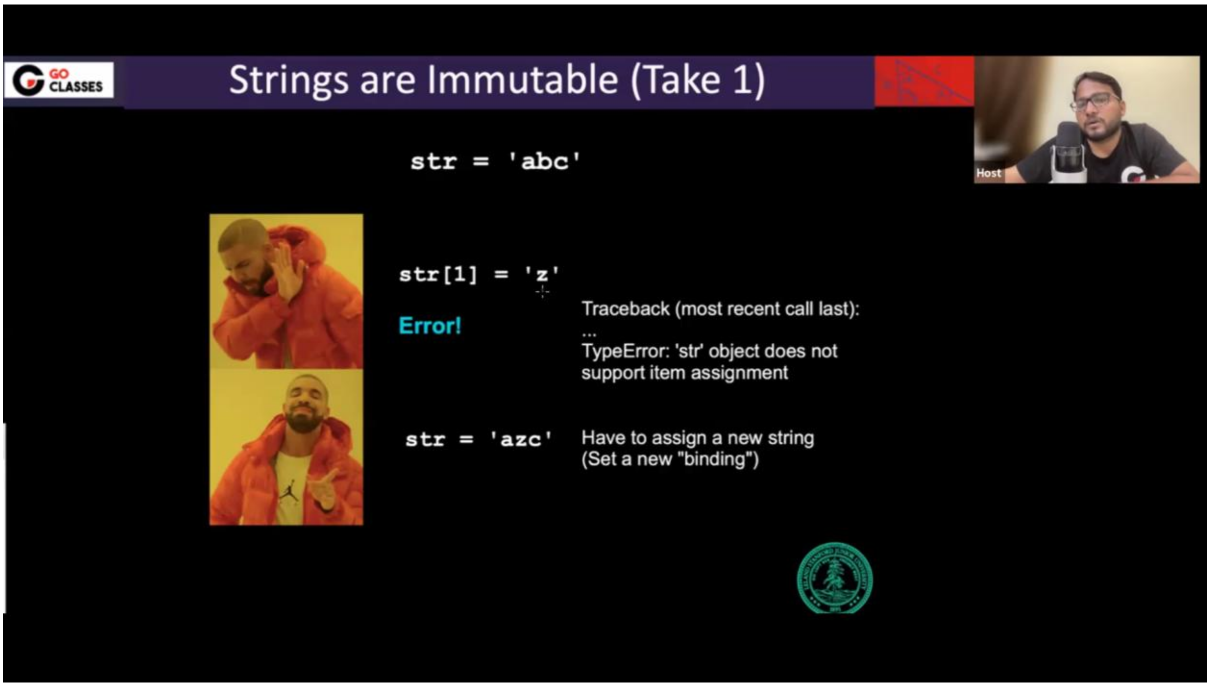
```
text = "hello!"
```

www.goclasses.in



Strings are Immutable

- Python strings are ***immutable***: once a string has been created **you cannot set characters**.
- To change a string:
  - ***Create a new string*** holding the new value you want it to have via concatenation.
  - Can *reassign* to the same string variable.
- ***Important consequence:*** if you pass a string into a function, you are guaranteed that string won't be changed.
  - Similar to behavior of int and float when passed to a function



Strings are Immutable (Take 1)

```
str = 'abc'
```

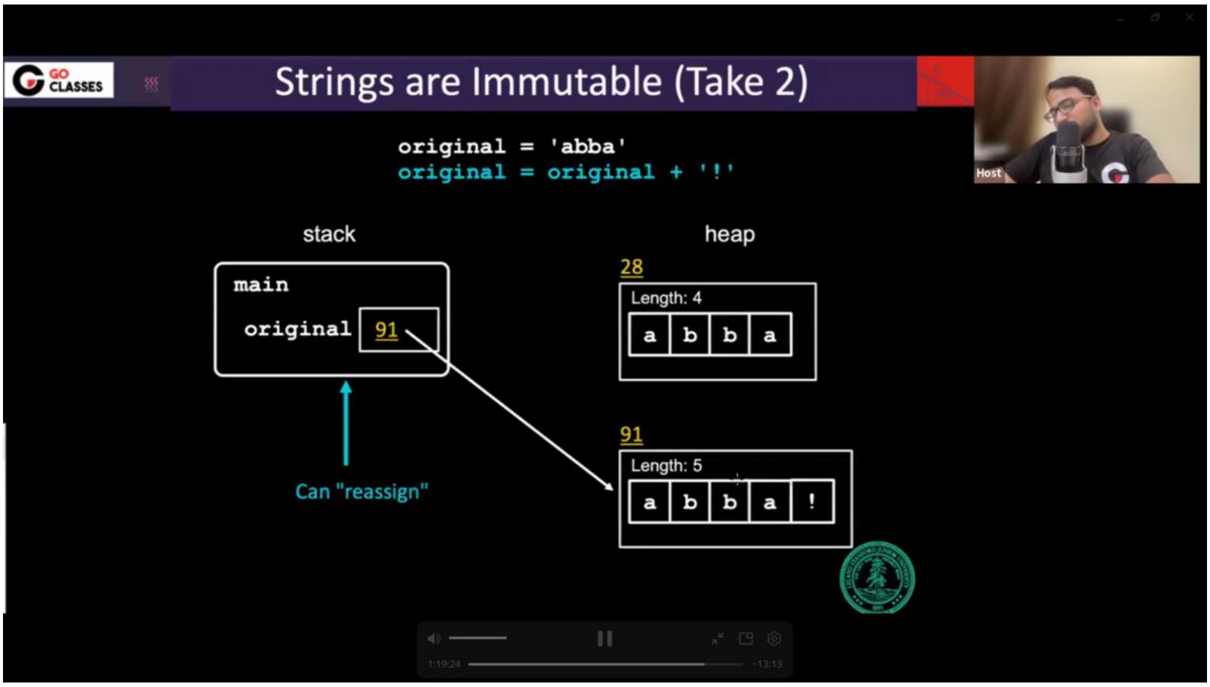
```
str[1] = 'z'
```

Error!

```
Traceback (most recent call last):
...
TypeError: 'str' object does not
support item assignment
```

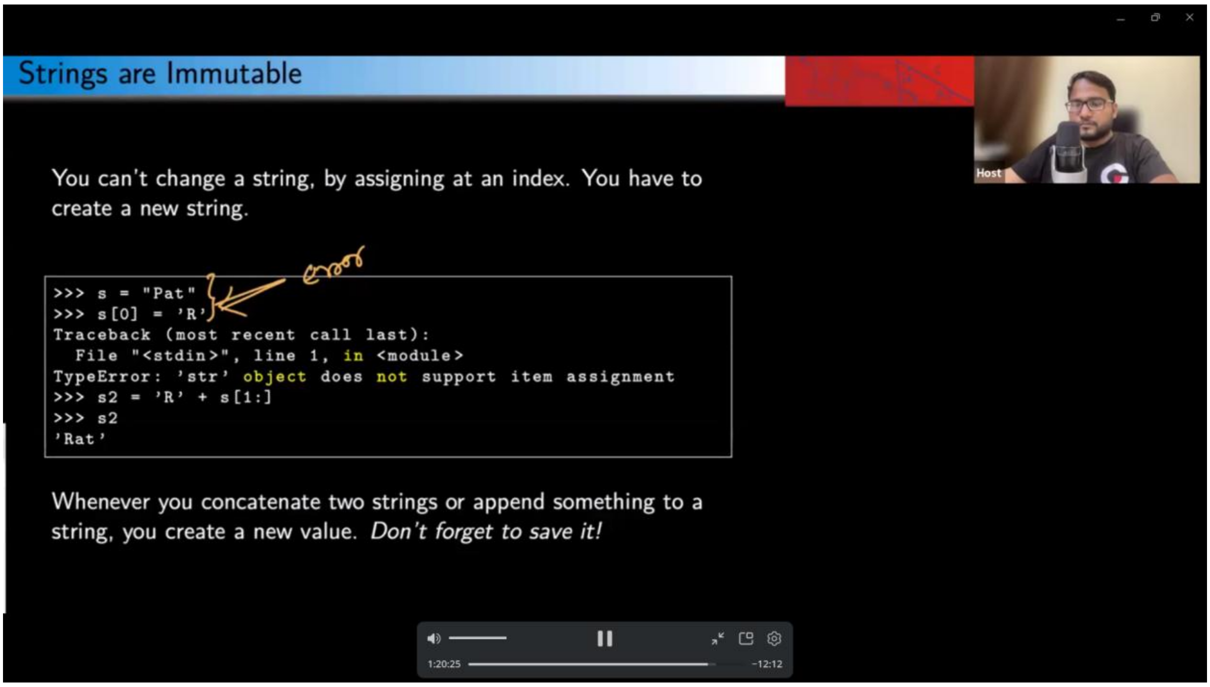
```
str = 'azc'
```

Have to assign a new string
(Set a new "binding")



Strings are Immutable (Take 2)

```
original = 'abba'
original = original + '!'
```



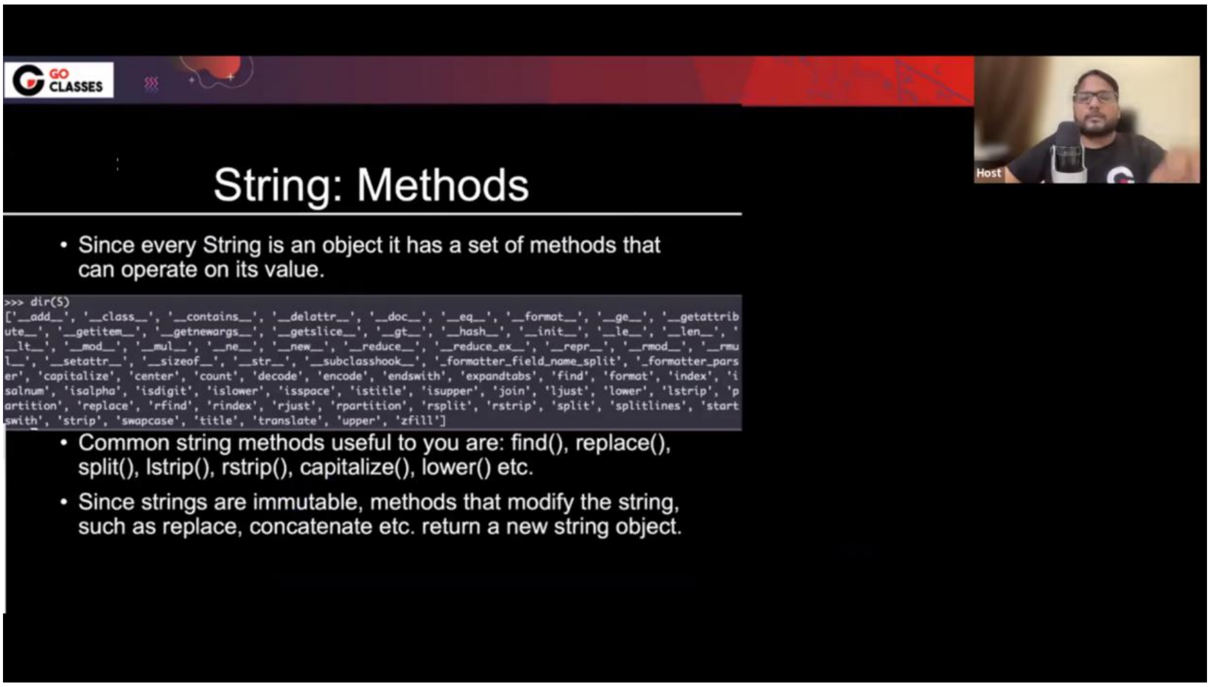
Strings are Immutable

You can't change a string, by assigning at an index. You have to create a new string.

```
>>> s = "Pat"
>>> s[0] = 'R'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> s2 = 'R' + s[1:]
>>> s2
'Rat'
```

error

Whenever you concatenate two strings or append something to a string, you create a new value. *Don't forget to save it!*

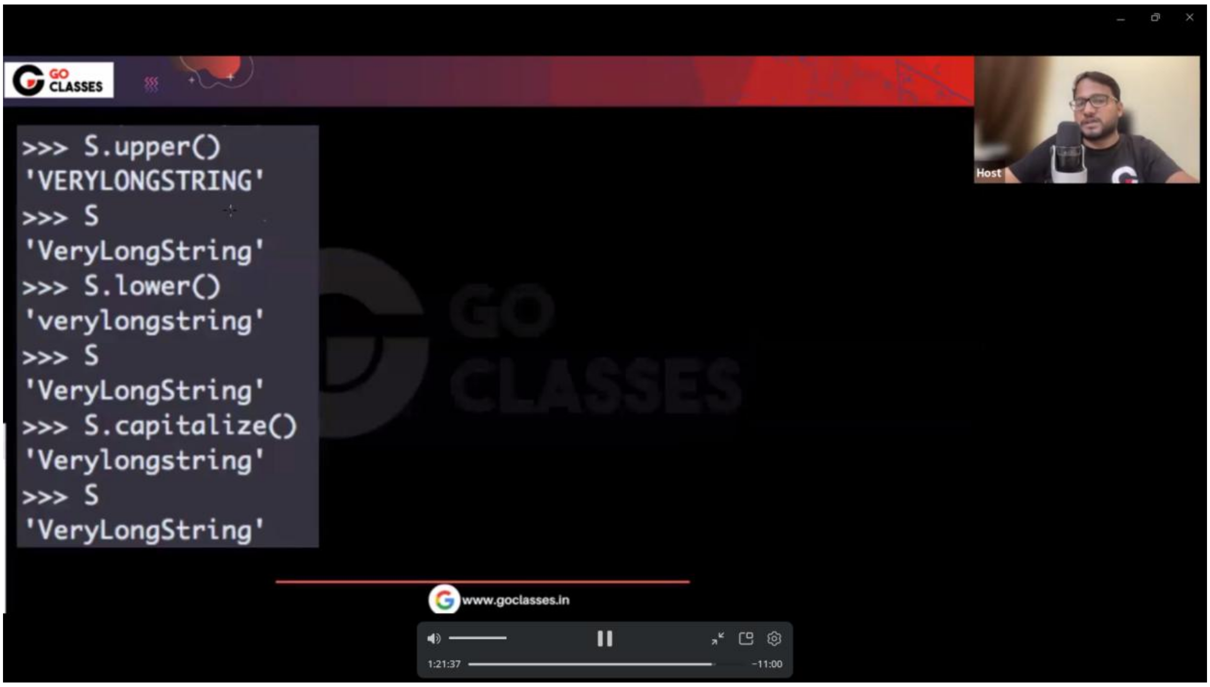


String: Methods

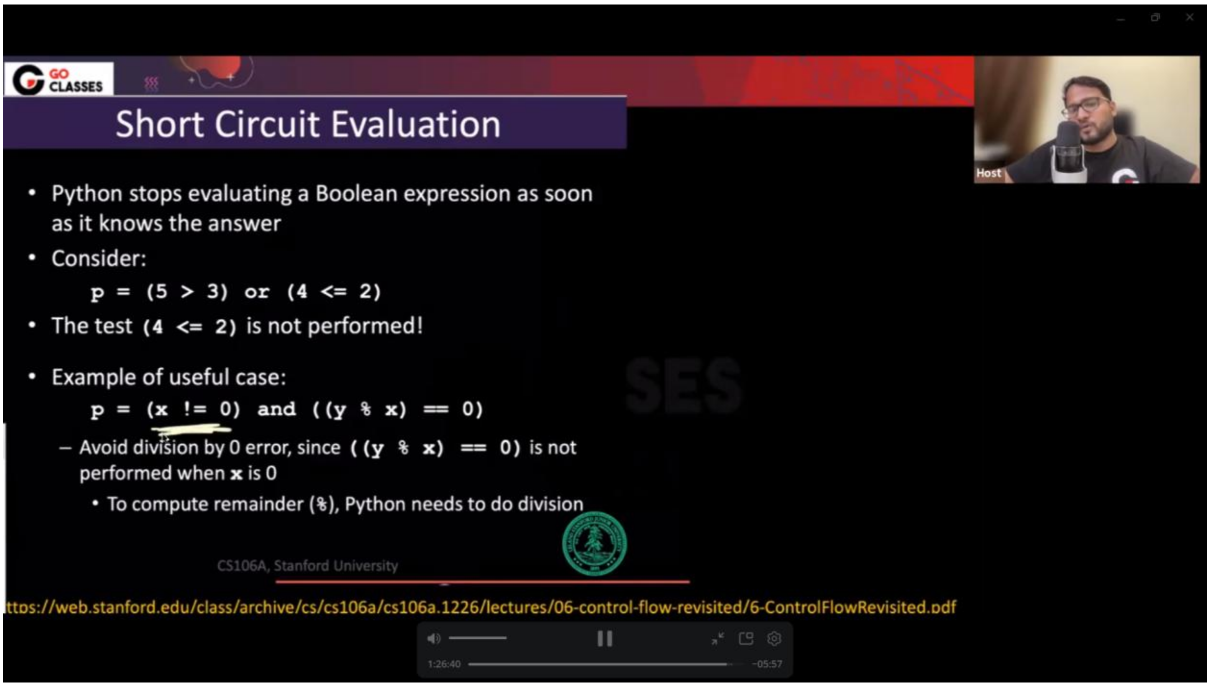
- Since every String is an object it has a set of methods that can operate on its value.

```
>>> dir(S)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattrib
ute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '
__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmu
l__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_formatter_field_name_split', '_formatter_pars
er', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index', 'i
salnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'p
artition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'start
swith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

- Common string methods useful to you are: find(), replace(), split(), lstrip(), rstrip(), capitalize(), lower() etc.
- Since strings are immutable, methods that modify the string, such as replace, concatenate etc. return a new string object.



```
>>> S.upper()
'VERYLONGSTRING'
>>> S
'VeryLongString'
>>> S.lower()
'verylongstring'
>>> S
'VeryLongString'
>>> S.capitalize()
'Verylongstring'
>>> S
'VeryLongString'
```



Short Circuit Evaluation

- Python stops evaluating a Boolean expression as soon as it knows the answer
- Consider:

```
p = (5 > 3) or (4 <= 2)
```

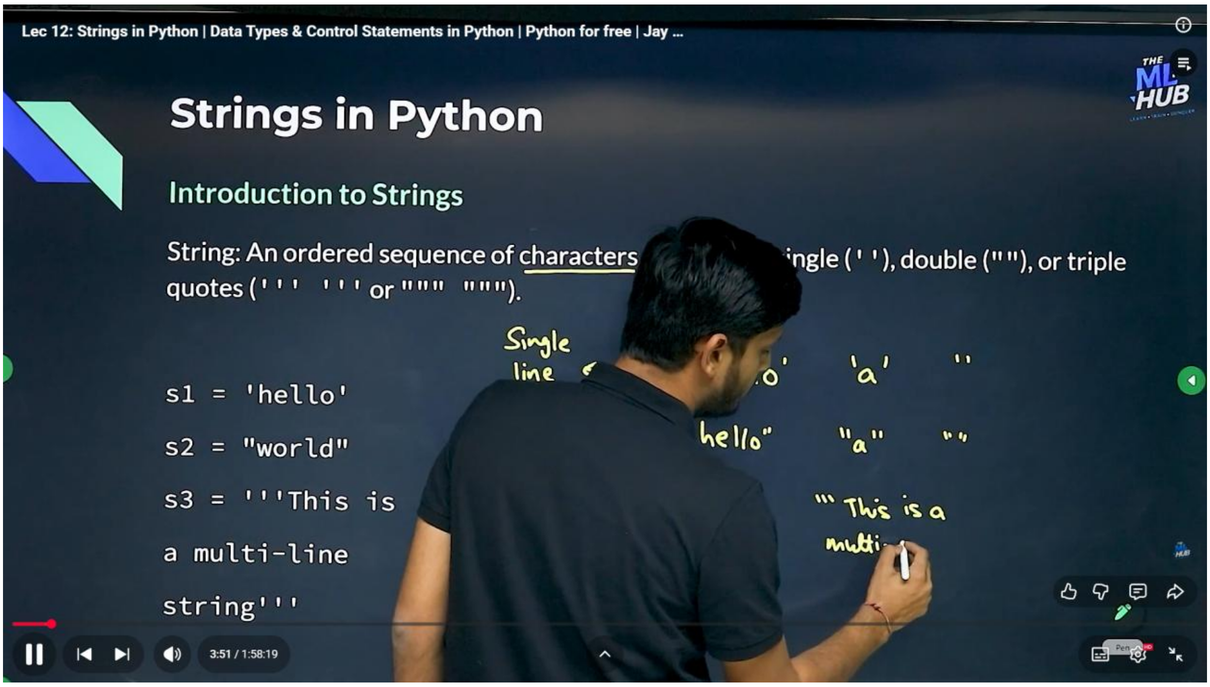
- The test `(4 <= 2)` is not performed!
- Example of useful case:

```
p = (x != 0) and ((y % x) == 0)
```

  - Avoid division by 0 error, since `((y % x) == 0)` is not performed when **x** is 0
    - To compute remainder (%), Python needs to do division

CS106A, Stanford University

ttps://web.stanford.edu/class/archive/cs/cs106a/cs106a.1226/lectures/06-control-flow-revisited/6-ControlFlowRevisited.pdf



Lec 12: Strings in Python | Data Types & Control Statements in Python | Python for free | Jay ...
THE ML HUB
3:51 / 1:58:19

Strings in Python

Introduction to Strings

String: An ordered sequence of characters ingle (' '), double (" "), or triple quotes (''' ''' or """ """).

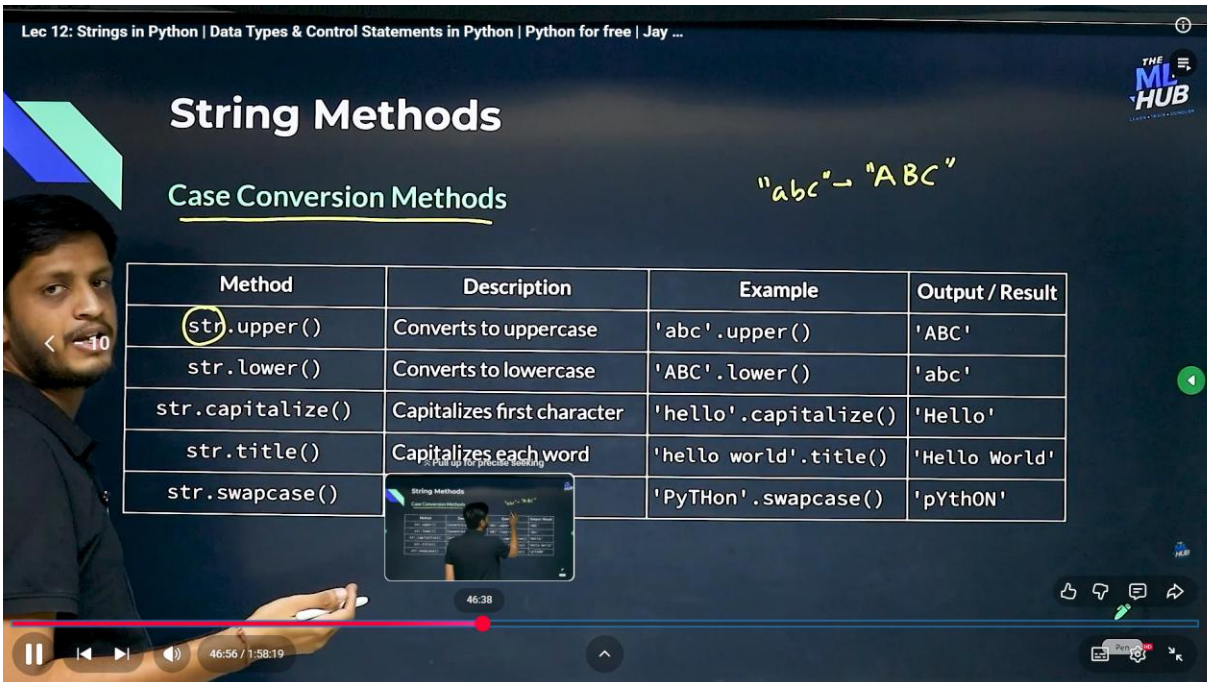
```
s1 = 'hello'

s2 = "world"

s3 = '''This is
a multi-line
string'''
```

Single line

o' 'a' ''

hello" "a" ""

''' This is a multi



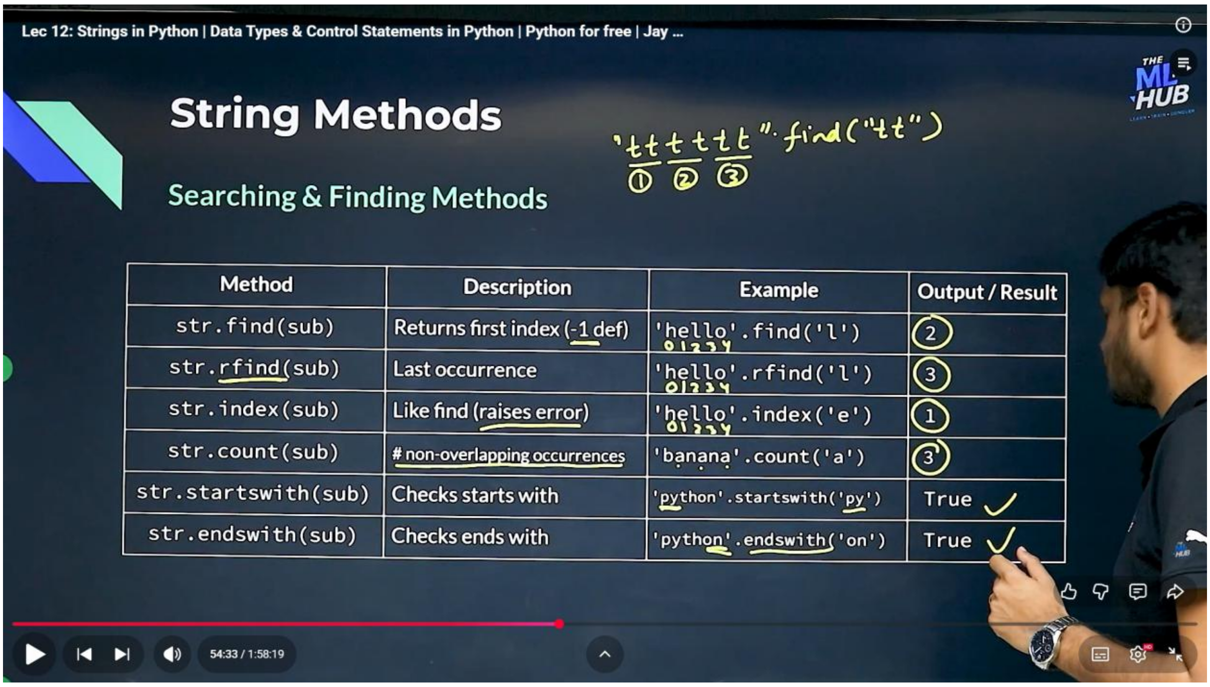
Lec 12: Strings in Python | Data Types & Control Statements in Python | Python for free | Jay ...

String Methods

Case Conversion Methods

"abc" → "ABC"

Method	Description	Example	Output / Result
`str.upper()`	Converts to uppercase	`'abc'.upper()`	`'ABC'`
`str.lower()`	Converts to lowercase	`'ABC'.lower()`	`'abc'`
`str.capitalize()`	Capitalizes first character	`'hello'.capitalize()`	`'Hello'`
`str.title()`	Capitalizes each word	`'hello world'.title()`	`'Hello World'`
`str.swapcase()`		`'PyTHon'.swapcase()`	`'pYthON'`



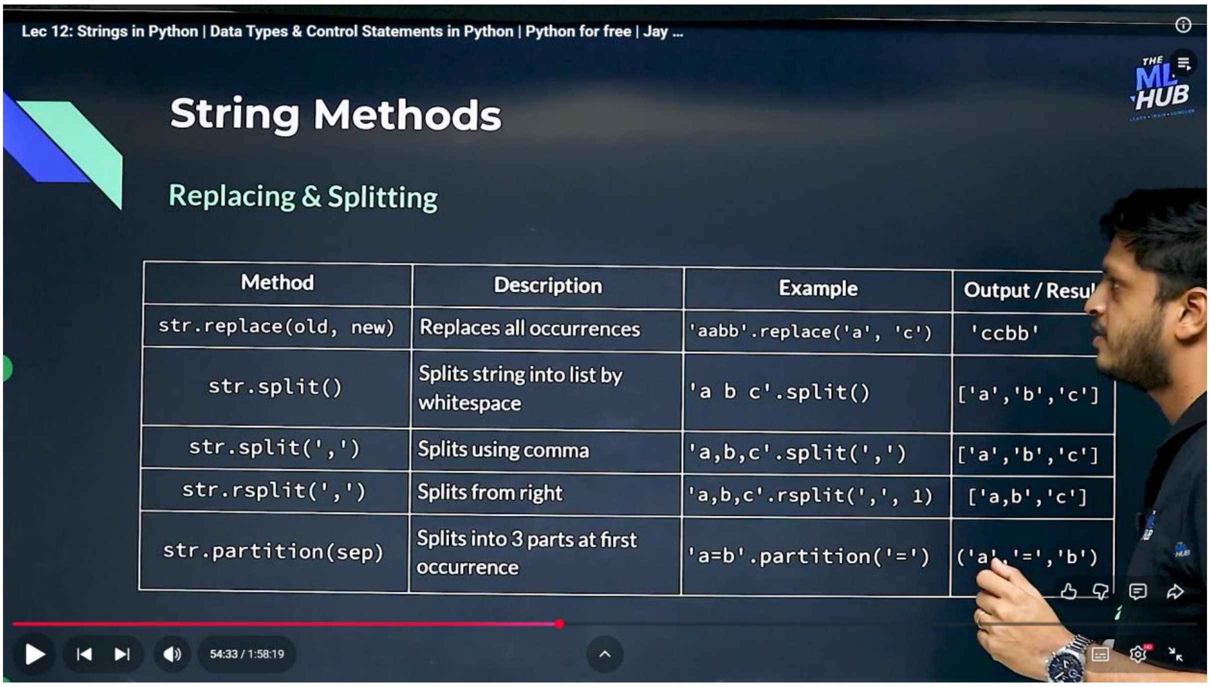
Lec 12: Strings in Python | Data Types & Control Statements in Python | Python for free | Jay ...

String Methods

Searching & Finding Methods

'tttttt".find("tt")
① ② ③

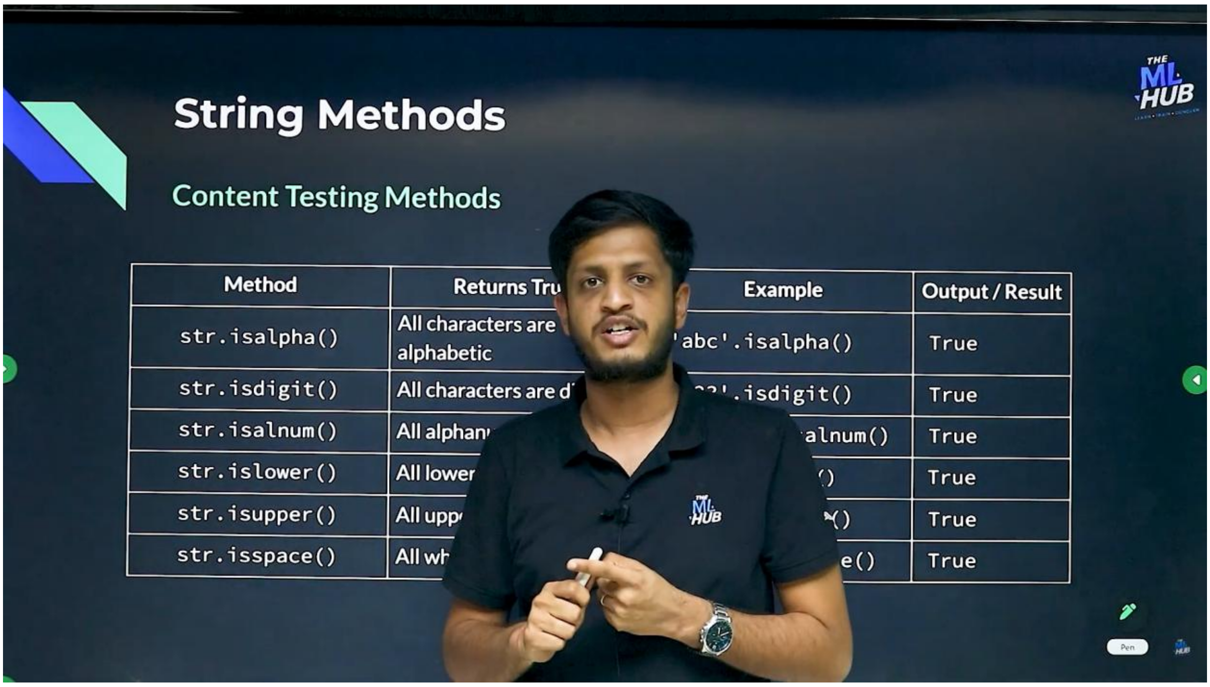
Method	Description	Example	Output / Result
`str.find(sub)`	Returns first index (-1 def)	`'hello'.find('l')` 01234	2
`str.rfind(sub)`	Last occurrence	`'hello'.rfind('l')` 01234	3
`str.index(sub)`	Like find (raises error)	`'hello'.index('e')` 01234	1
`str.count(sub)`	# non-overlapping occurrences	`'banana'.count('a')`	3
`str.startswith(sub)`	Checks starts with	`'python'.startswith('py')`	True ✓
`str.endswith(sub)`	Checks ends with	`'python'.endswith('on')`	True ✓


Lec 12: Strings in Python | Data Types & Control Statements in Python | Python for free | Jay ...
THE ML HUB
54:33 / 1:58:19

String Methods

Replacing & Splitting

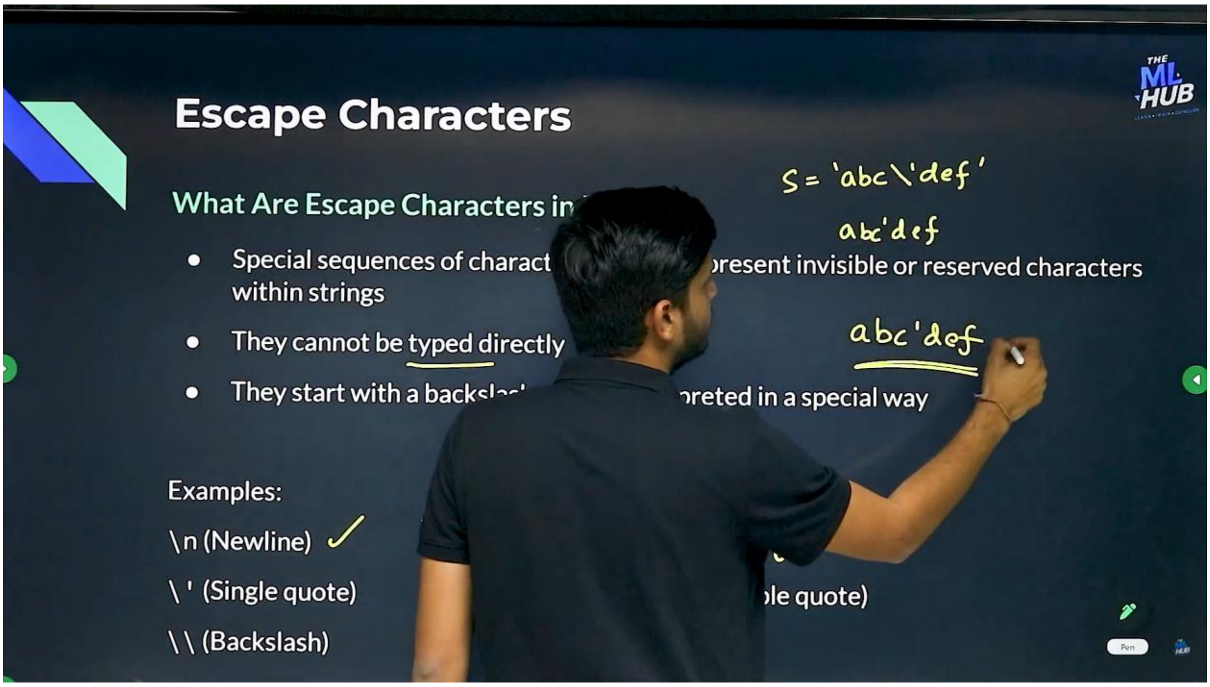
Method	Description	Example	Output / Resul
`str.replace(old, new)`	Replaces all occurrences	`'aabb'.replace('a', 'c')`	`'ccbb'`
`str.split()`	Splits string into list by whitespace	`'a b c'.split()`	`['a','b','c']`
`str.split(',')`	Splits using comma	`'a,b,c'.split(',')`	`['a','b','c']`
`str.rsplit(',')`	Splits from right	`'a,b,c'.rsplit(',', 1)`	`['a,b','c']`
`str.partition(sep)`	Splits into 3 parts at first occurrence	`'a=b'.partition('=')`	`('a','=','b')`



String Methods

Content Testing Methods

Method	Returns Tru	Example	Output / Result
`str.isalpha()`	All characters are alphabetic	`'abc'.isalpha()`	True
`str.isdigit()`	All characters are d	`'.isdigit()`	True
`str.isalnum()`	All alphan	`alnum()`	True
`str.islower()`	All lower	`()`	True
`str.isupper()`	All upp	`()`	True
`str.isspace()`	All wh	`e()`	True



Escape Characters

What Are Escape Characters in

- Special sequences of charact present invisible or reserved characters within strings
- They cannot be typed directly
- They start with a backsla preted in a special way

Examples:

\n (Newline) ✓

\ ' (Single quote)

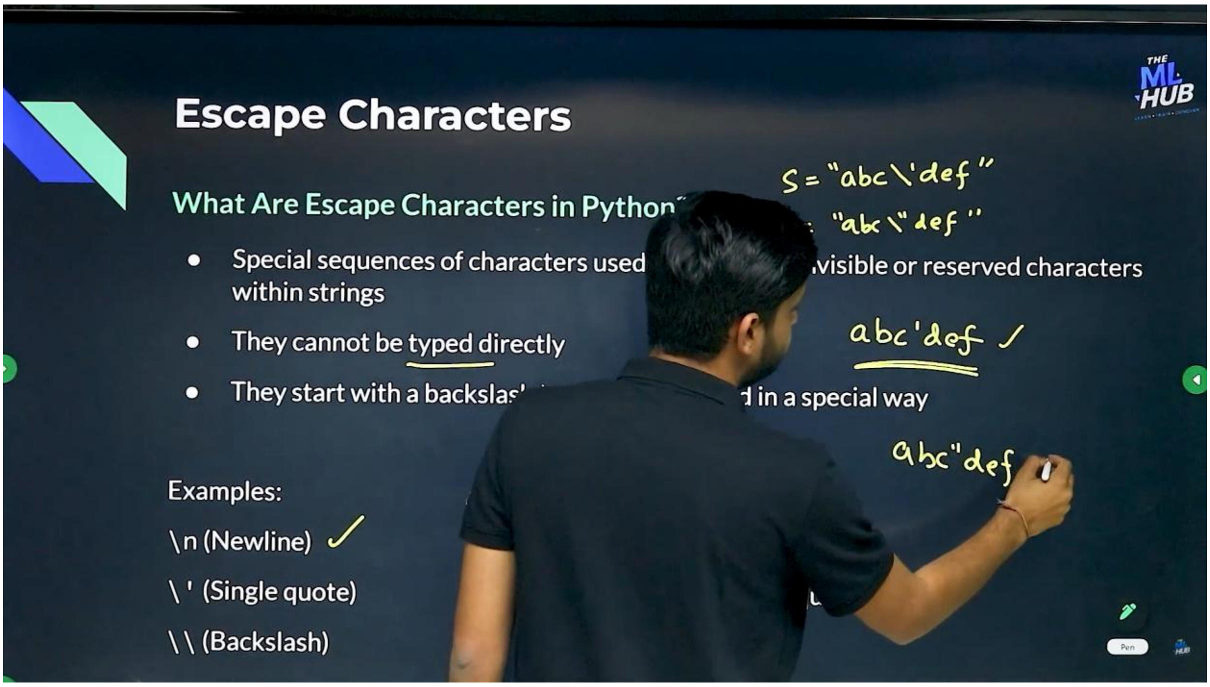
\ \ (Backslash)

ble quote)

S = 'abc\'def'

abc'def

abc'def



Escape Characters

What Are Escape Characters in Python

- Special sequences of characters used visible or reserved characters within strings
- They cannot be typed directly
- They start with a backslas d in a special way

Examples:

\n (Newline) ✓

\ ' (Single quote)

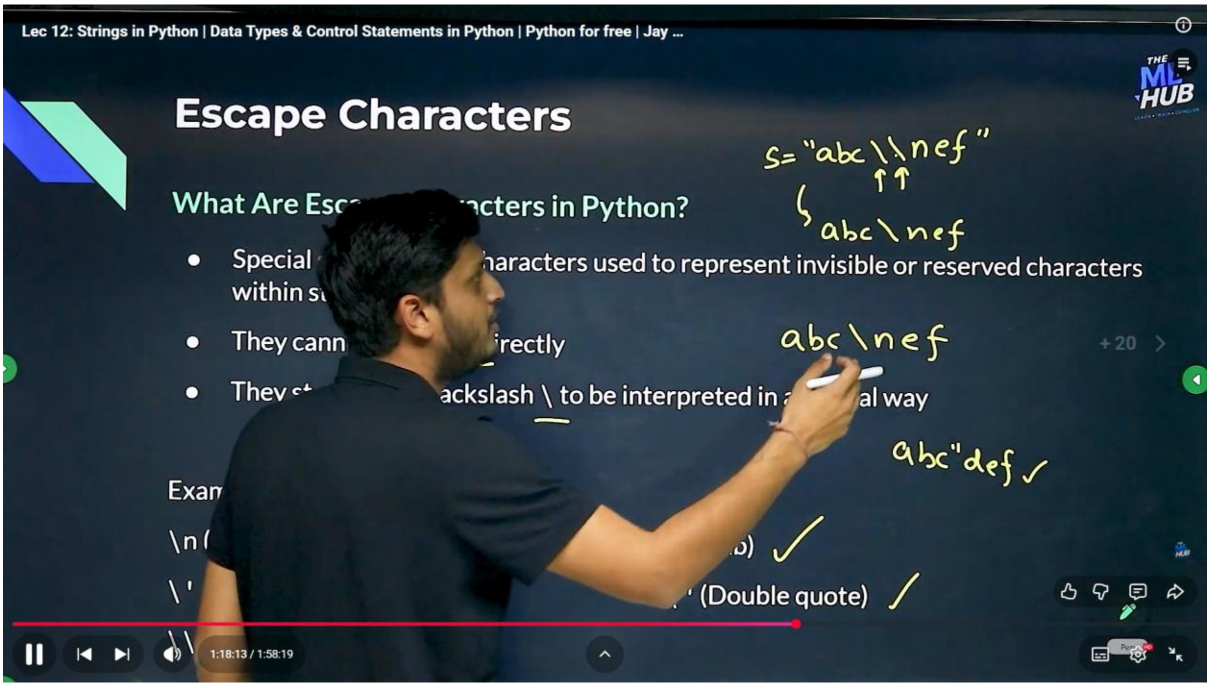
\ \ (Backslash)

S = "abc\'def"

"abc\"def"

abc'def ✓

abc"def


Lec 12: Strings in Python | Data Types & Control Statements in Python | Python for free | Jay ...

Escape Characters

What Are Esc acters in Python?

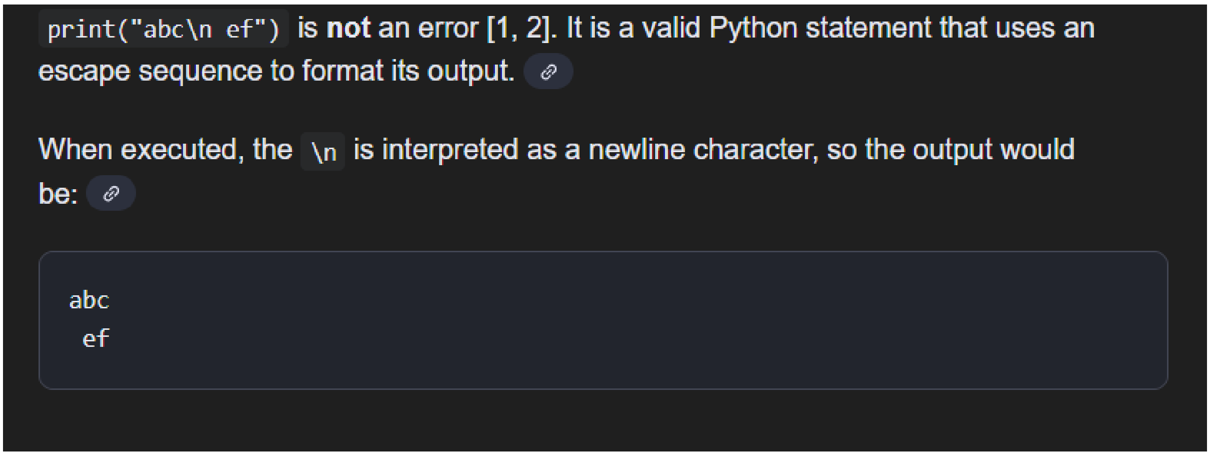
- Special haracters used to represent invisible or reserved characters within s
- They cann irectly
- They s ackslash \ to be interpreted in a al way

Exan

\n (b) ✓

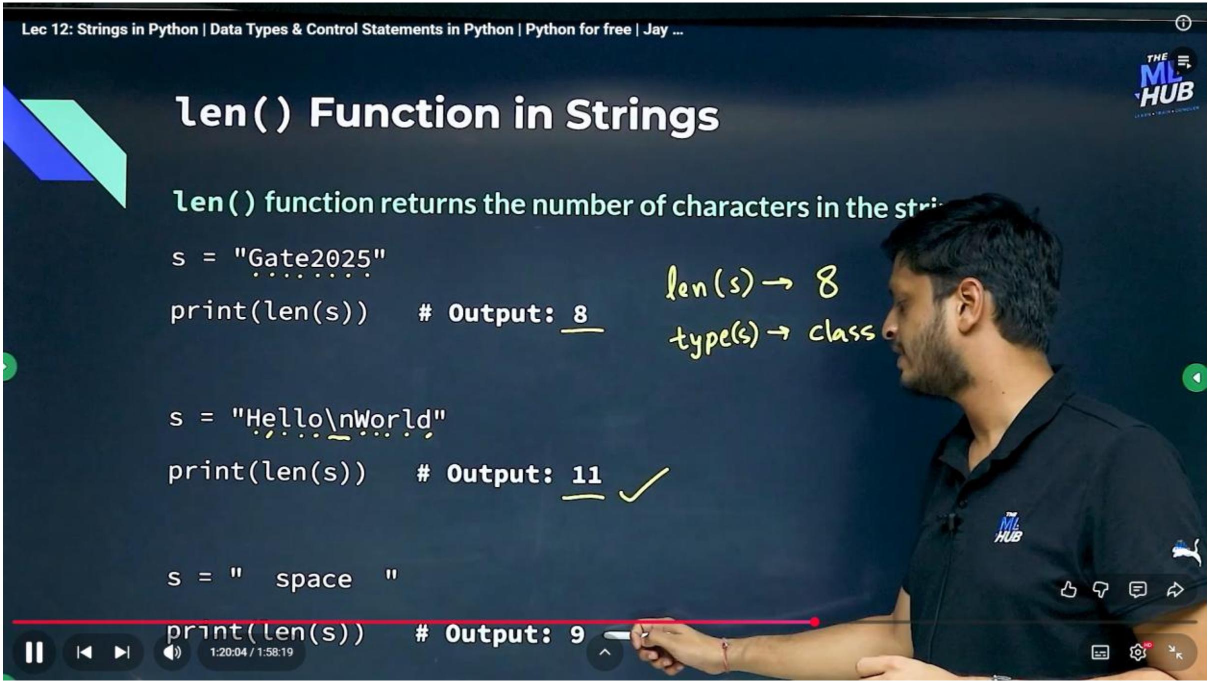
\' ' (Double quote) ✓

\\


`print("abc\n ef")` is **not** an error [1, 2]. It is a valid Python statement that uses an escape sequence to format its output.

When executed, the `\n` is interpreted as a newline character, so the output would be:

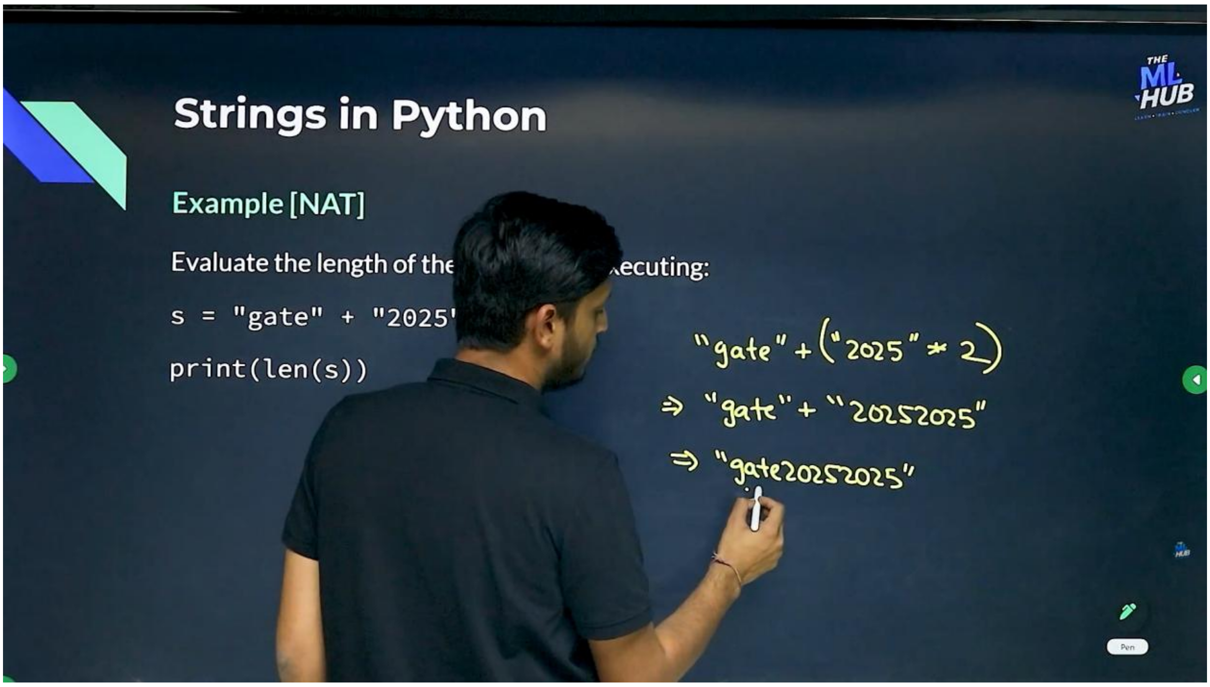
```
abc
 ef
```


Lec 12: Strings in Python | Data Types & Control Statements in Python | Python for free | Jay ...

len() Function in Strings

len() function returns the number of characters in the str

```
s = "Gate2025"
print(len(s))    # Output: 8

s = "Hello\nWorld"
print(len(s))    # Output: 11

s = "  space  "
print(len(s))    # Output: 9
```



Strings in Python

Example [NAT]

Evaluate the length of the xecuting:

```
s = "gate" + "2025'
print(len(s))
```

"gate" + ("2025" * 2)

⇒ "gate" + "20252025"

⇒ "gate20252025"