

The screenshot shows a video player interface for a Go Classes lesson. The title "Strings are Made of Characters" is displayed at the top. A video frame on the right shows a host wearing glasses and a black t-shirt. Below the title, a text block asks: "Unlike numbers and Booleans, strings can be broken down into individual parts (characters). How can we access a specific character in a string?" A diagram below illustrates this with the word "STELLA" in a box, followed by six smaller boxes labeled 0 through 5, each containing one of the letters S, T, E, L, L, and A respectively.

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

0 1 2 3 4 5

<https://www.cs.cmu.edu/~15110/slides/week4-2-string.pdf>

The screenshot shows a Jupyter Notebook interface. The URL bar indicates the notebook is running on localhost:8888. The notebook title is "Python Programming GO Classes". A code cell has been run, showing the output of "STELLA"[0] as 'S'. Below the cell, there are several empty input fields labeled "In []:" for further code execution. At the bottom of the screen, a video player controls are visible, showing a timestamp of 02:20 and a duration of -1:30:17.

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

The screenshot shows a video player interface. At the top left is the 'GO CLASSES' logo. On the right, there is a small video frame showing a man with glasses and a beard, identified as the 'Host'. The main title 'Negative Indexing' is displayed in white text on a dark red bar at the top. Below the title, the string 'STRING = "AASHINA"' is shown. A diagram illustrates indexing for this string. It features two rows of boxes: 'REVERSE INDEX' (indices -7 to -1) and 'FORWARD INDEX' (indices 0 to 6). The string 'AASHINA' is split into two parts: 'A A S H I N A' above a horizontal line and 'E S' below it. The reverse index row has boxes at indices -7, -6, -5, -4, -3, -2, and -1. The forward index row has boxes at indices 0, 1, 2, 3, 4, 5, and 6. The letter 'A' at index 0 is highlighted with a yellow box. The URL 'www.goclasses.in' is visible at the bottom of the screen.

The screenshot shows a video player interface. At the top left is the 'GO CLASSES' logo. On the right, there is a small video frame showing a man with glasses and a beard, identified as the 'Host'. The main title 'String Length' is displayed in large white text at the top. Below the title, there is a bulleted list of points: '• You can find the length of a string using 'len()'' followed by three examples: '• len("William") – gives 7', '• len("") – gives 0'. A large watermark 'ES' is visible across the center of the screen. The URL 'www.goclasses.in' is visible at the bottom of the screen.

GO CLASSES

Strings are often made through concatenation

Host

```
s1 = "CS106"
s2 = "A"
s3 = "I got an " + s2 + " in " + s1 + s2
    ↓
print(s3)
```

I got an A in CS106A

www.goclasses.in

11:56 - 1:20:41

GO CLASSES

Strings are often made through concatenation

Host

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

I got an A in CS106A

In [16]: s1 = "GO"
s2 = "Classes"
s1+s2
Out[16]: 'GOClasses'

www.goclasses.in

12:42 - 1:19:55

The screenshot shows a video player interface for 'GO CLASSES'. The title 'String Concatenation' is displayed at the top. A video frame on the right shows a host wearing glasses and a black t-shirt with a 'G' logo. Below the video frame, the word 'Host' is written. The video content includes a bulleted list and some code examples:

- 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

At the bottom of the video frame, there is a link: <https://teachinglondoncomputing.org/wp-content/uploads/2014/10/topic1-2-v4.pdf>. The video player has a progress bar showing 12:52 and a total duration of 1:19:45.

The screenshot shows a video player interface for 'GO CLASSES'. The title 'Repetition' is displayed at the top. A video frame on the right shows a host wearing glasses and a black t-shirt with a 'G' logo. Below the video frame, the word 'Host' is written. The video content includes a text explanation and some code examples:

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

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

At the bottom of the video frame, there is a watermark for 'www.goclasses.in'. The video player has a progress bar showing 13:01 and a total duration of 1:19:36.

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

www.goclasses.in

14:15 - 1:18:22

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
```

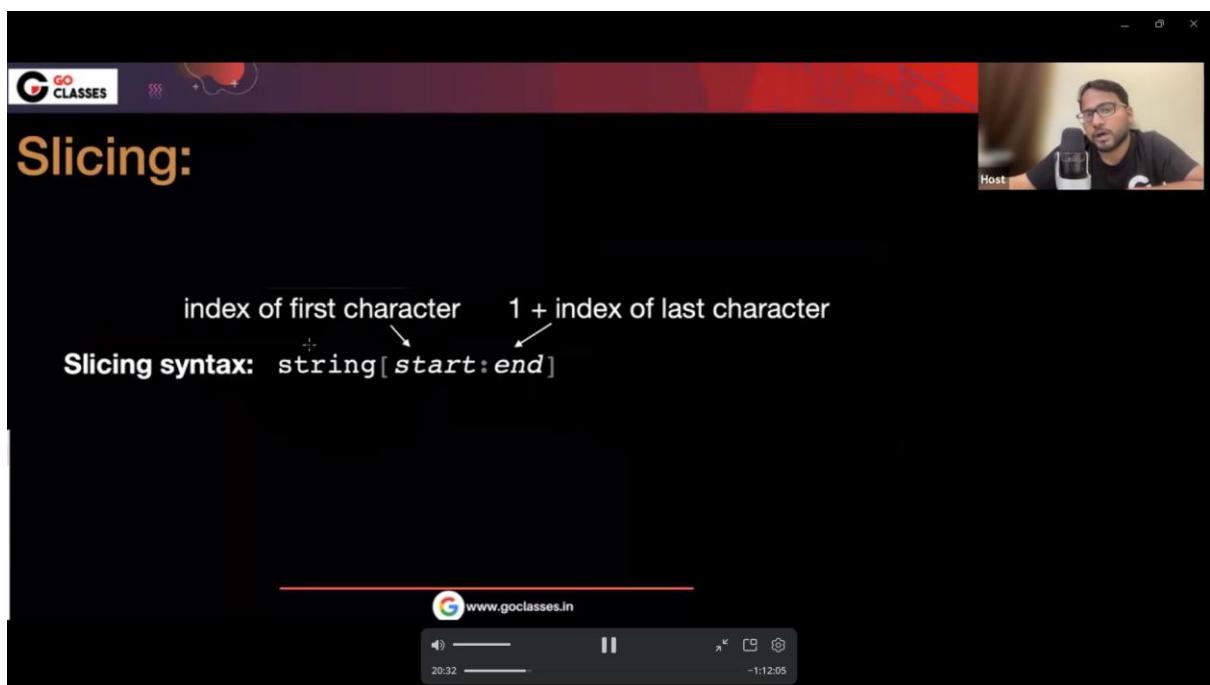
```
In [19]: "G" in "GO"  
Out[19]: True  
  
In [21]: "L" not in "GO"  
Out[21]: True
```

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

A screenshot of a video player interface for 'GO CLASSES'. In the top right corner, there is a video feed of a man wearing glasses and a black t-shirt, identified as the 'Host'. The video player has standard controls like play/pause, volume, and a progress bar showing 17:07 of a 1:15:30 video. The main content area features a large watermark for 'GO CLASSES' in the center. On the left side of the screen, there is a Python code editor window displaying the following code:

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

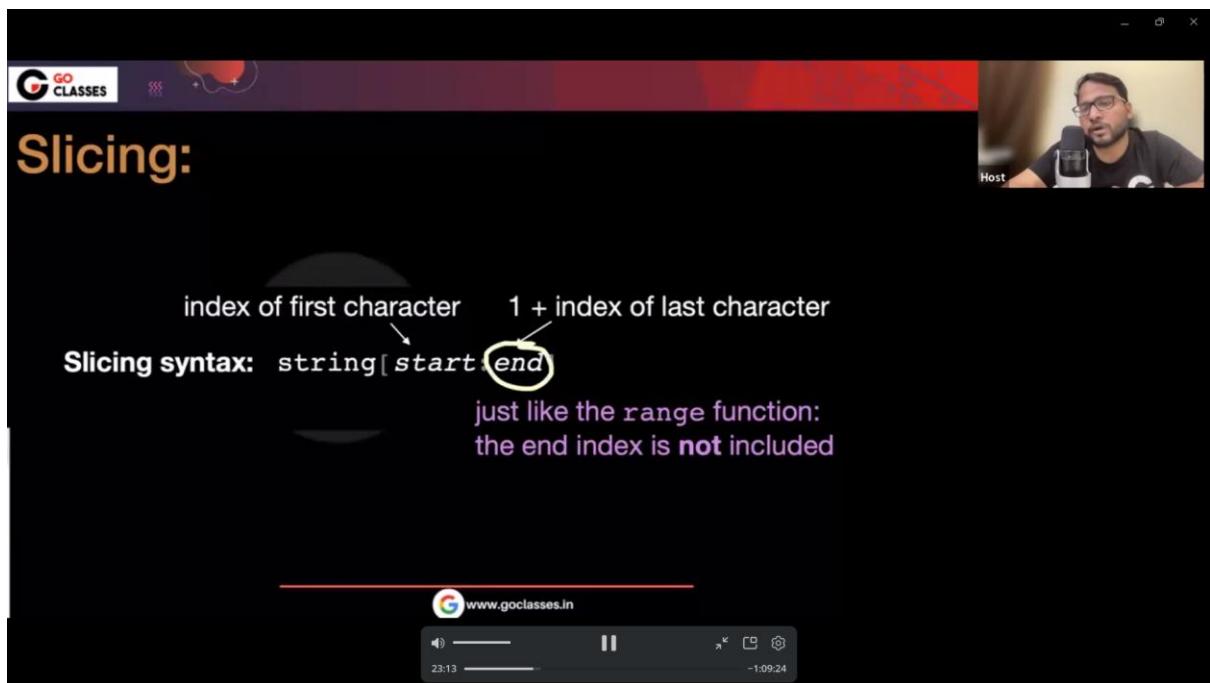
A screenshot of a video player interface for 'GO CLASSES', identical to the one above but at a later timestamp. The host is still visible in the top right. The video player controls show 17:34 of a 1:15:03 video. The main content area shows the same Python code as the previous screenshot, demonstrating string membership and non-membership operations.



A screenshot of a video player interface for 'GO CLASSES'. The video frame shows a man speaking. In the top left corner is the 'GO CLASSES' logo. The main content area displays a code example in a Jupyter Notebook cell:

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

Below the code cell, there is a diagram illustrating string slicing. It shows a string `s = "Hello"` with indices 0, 1, 2, 3, and 4 above the characters H, e, l, l, o respectively. Below this, the expression `s[0:3]` is shown with an arrow pointing to the result `'Hel'`. A large watermark for 'GO CLASSES' is visible across the middle of the screen.



Host

```
alph = "abcdefghijkl"  
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] # => "abcdefghijkl"
```

www.goclasses.in

23:50 - 1:08:47

Host

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

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

www.goclasses.in

24:34 - 1:08:03

The screenshot shows a video player interface for 'GO CLASSES'. In the top right corner, there is a small video thumbnail of a man with glasses and a beard, labeled 'Host'. The main content area displays a diagram explaining the syntax of Python string slicing. The diagram includes the following text and annotations:

Slicing syntax: `string[start:end]`

- index of first character
- 1 + index of last character
- If omitted, `start` defaults to 0
- If omitted, `end` defaults to `len(string)`

Below the diagram is a watermark for 'www.goclasses.in'.

The screenshot shows a Go Classes video player interface. At the top left is the Go Classes logo. On the right is a video feed of a host wearing glasses and a black t-shirt, with the word "Host" below it. The main area contains a Jupyter Notebook interface with several code cells:

- 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'

Below the notebook is a media control bar with volume, play/pause, and other controls. The video player has a progress bar from 28:36 to -1:04:01.

The screenshot shows a Go Classes video player interface. At the top left is the Go Classes logo. On the right is a video feed of a host wearing glasses and a black t-shirt, with the word "Host" below it. The main area contains a diagram and a Jupyter Notebook interface:

[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 '

Below the notebook is a media control bar with volume, play/pause, and other controls. The video player has a progress bar from 30:38 to -1:01:59.

• 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' ←
```

Host

```
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

Host

GO CLASSES

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] → in
B. last_name[6:-1] → in
C. last_name[-3:] → in
D. last_name[-2:8] → in

www.goclasses.in

36:21 -56:16

Host

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.

GO CLASSES

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

www.goclasses.in

37:44 -54:53

Host

A screenshot of a video player interface for 'GO CLASSES'. The video frame shows a male host wearing glasses and a black t-shirt, sitting in front of a microphone. The video player has a dark theme with a red header bar. In the top right corner of the video frame, the word 'Host' is visible. Below the video frame, there is a Python code editor window displaying the following code:

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

Below the code, a character map for the string 'Williamstown' is shown, with indices from 0 to 11 above the letters and indices from -12 to -1 below the letters. The letter 'W' is at index 0, 'i' is at index 1, 'l' is at index 2, 'l' is at index 3, 'l' is at index 4, 'i' is at index 5, 'a' is at index 6, 'm' is at index 7, 's' is at index 8, 't' is at index 9, 'o' is at index 10, and 'n' is at index 11. Indices from -12 to -1 are shown below the letters, with 'n' at index -1 and 'W' at index -12.

The video player interface includes a progress bar at the bottom with '38:03' on the left and '-54:34' on the right, and a control bar with icons for volume, play/pause, and other media controls.

A screenshot of a video player interface for 'GO CLASSES', continuing from the previous frame. The host is still visible in the video frame. The video player interface is identical to the first one, with a dark theme and a red header bar. The Python code editor window now displays the following code:

```
>>> 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'
```

Below the code, a character map for the string 'Williamstown' is shown, with indices from 0 to 11 above the letters and indices from -12 to -1 below the letters. The letter 'W' is at index 0, 'i' is at index 1, 'l' is at index 2, 'l' is at index 3, 'i' is at index 4, 'm' is at index 5, 's' is at index 6, 't' is at index 7, 'o' is at index 8, 'n' is at index 9, 't' is at index 10, and 'w' is at index 11. Indices from -12 to -1 are shown below the letters, with 'w' at index -1 and 'W' at index -12.

The video player interface includes a progress bar at the bottom with '39:45' on the left and '-52:52' on the right, and a control bar with icons for volume, play/pause, and other media controls.

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"
```

www.goclasses.in

42:33 -> 50:04

negative steps

- Extracting from the *right end*

$s = "Hello"$

$s[4:0:-1]$ gives 'olle'

olle

Diagram illustrating string slicing with negative indices:

The string "Hello" is shown with indices 0 through 4 above it. Brackets indicate the slice from index 4 down to index 0. The characters are labeled as follows:

- Index 0: o
- Index 1: l
- Index 2: l
- Index 3: e
- Index 4: o

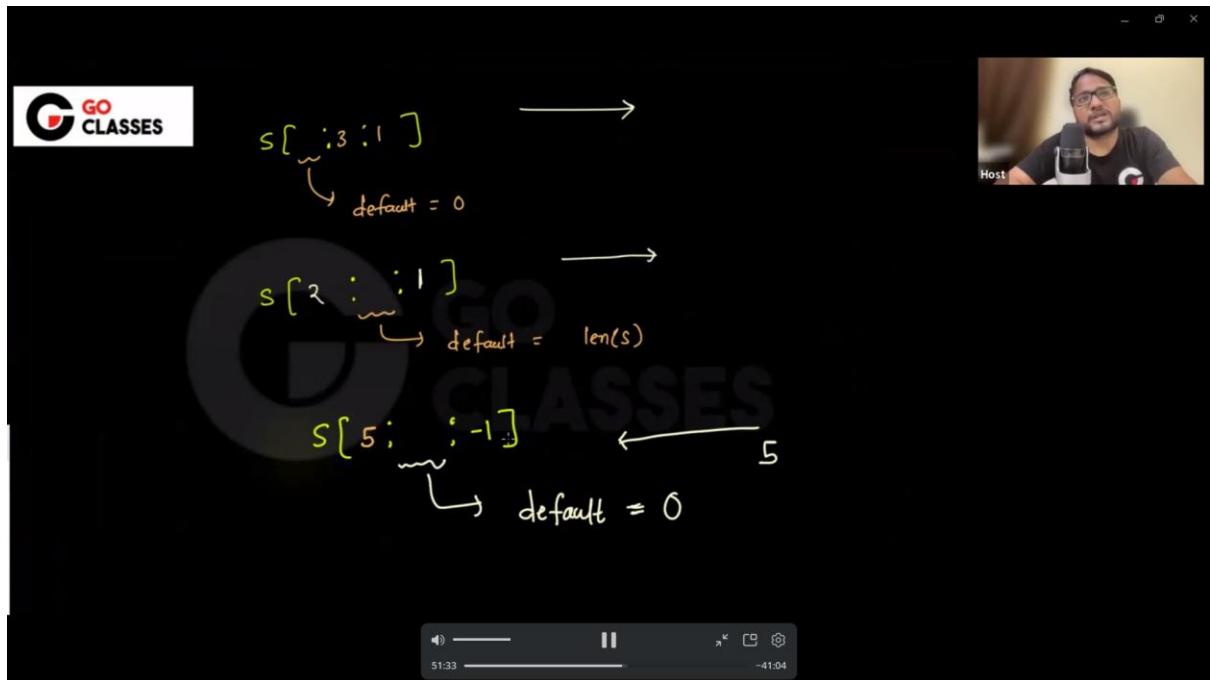
Brackets group the characters from index 4 down to index 0, resulting in the reversed substring 'olle'.

www.goclasses.in

44:03 -> 48:34

The screenshot shows a video player interface for 'GO CLASSES'. At the top, there's a navigation bar with icons for search, refresh, and other controls. On the right, a small video thumbnail of a host wearing glasses and a black t-shirt with a 'G' logo is visible, labeled 'Host'. Below the thumbnail, the video content starts with a diagram illustrating string slicing. A horizontal row of letters 'H E L L O W O R L D' is shown, with each letter in its own box. Above each letter is an upward-pointing arrow, and below each letter is text indicating its position relative to the start or end of the string. The positions are listed as '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', and '10 or -1'. Below this diagram, a note reads: '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.' A progress bar at the bottom of the video player shows the video is at 46:06 of 46:31.

The screenshot shows a video player interface for 'GO CLASSES'. At the top, there's a navigation bar with icons for search, refresh, and other controls. On the right, a small video thumbnail of a host wearing glasses and a black t-shirt with a 'G' logo is visible, labeled 'Host'. Below the thumbnail, the video content starts with the text 'Default values' in yellow. It then shows a code snippet: `s[:3:]`. Following this, there's a section titled 'String[a:b:c]' with two bullet points: '• c is negative you count backwards' and '• The default values of a and b depends on sign of c.' A progress bar at the bottom of the video player shows the video is at 48:15 of 44:22.



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`)

```
In [1]: string = "Howdy doody"
In [2]: string[::-1]
Out[2]: 'Howdy doody'

In [3]: string[::-1]
Out[3]: 'ydoood 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.
```

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]	(f) ALPHABET[1:-1]
(b) ALPHABET[-3:-1]	(g) ALPHABET[0:5:2]
(c) ALPHABET[:3]	(h) ALPHABET[::-1]
(d) ALPHABET[-1:]	(i) ALPHABET[5:2:-1]
(e) ALPHABET[14:-12]	(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!"
```

stack

main
 text 28

heap

28 →

Length: 6					
H	e	l	l	o	!

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'
```



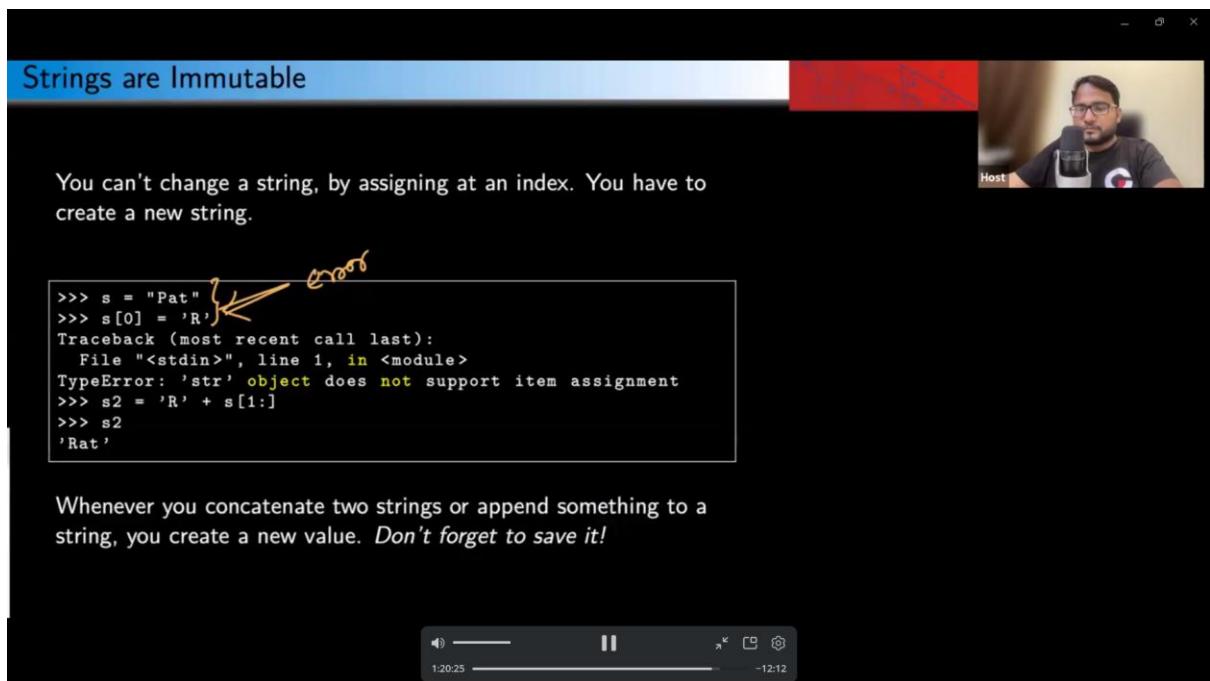
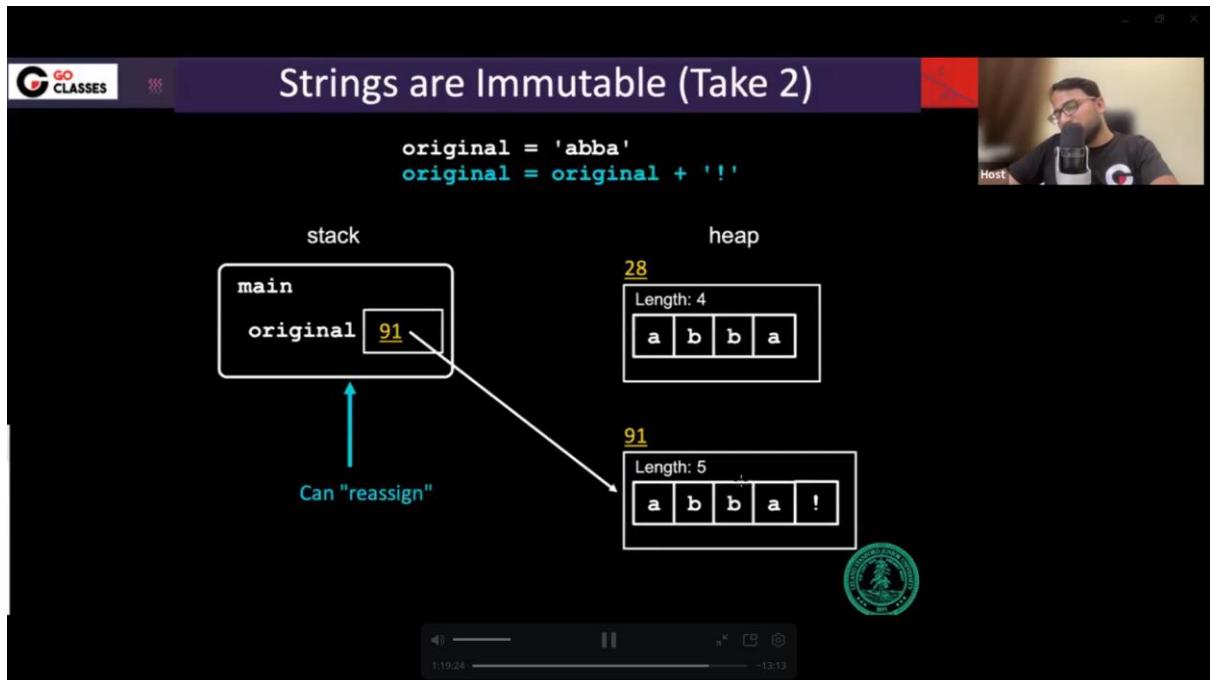
```
str[1] = 'z'  
Error!
```

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")





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__', '__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_formatter_field_name_split', '_formatter_parser', '_capitalize', '_center', '_count', '_decode', '_endswith', '_expandtabs', '_find', '_format', '_index', '_ljust', '_lstrip', '_lower', '_lstrip', '_partition', '_replace', '_rfind', '_rindex', '_rjust', '_rpartition', '_rsplit', '_rstrip', '_split', '_splitlines', '_startswith', '_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'
```

www.goclasses.in

1:21:37 - 11:00

The screenshot shows a video player interface. At the top left is the 'GO CLASSES' logo. The main title 'Short Circuit Evaluation' is displayed in a purple bar. On the right, there is a video feed of a host wearing glasses and a dark t-shirt, with the word 'Host' below it. The video content itself is a presentation slide with the following text:

- 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

Below the slide, the Stanford University seal is visible, followed by the text 'CS106A, Stanford University'. The video player controls at the bottom show a progress bar from 1:26:40 to -05:57.

The screenshot shows a video player interface. The top bar includes the text 'Lec 12: Strings in Python | Data Types & Control Statements in Python | Python for free | Jay ...' and the 'THE ML HUB' logo. The main content area has a dark background with a blue and green geometric graphic on the left. The title 'Strings in Python' is centered at the top. Below it, the section 'Introduction to Strings' is introduced. The video content shows a person writing code on a chalkboard. The code examples are:

```
s1 = 'hello'  
s2 = "world"  
s3 = '''This is  
a multi-line  
string'''
```

On the chalkboard, the person is writing the following text:

Single line
'hello' 'a'
"hello" "a" "
''' This is a
multi-line
string'''

At the bottom of the screen, video player controls show a progress bar from 3:51 / 1:58:19.

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
<code>str.upper()</code>	Converts to uppercase	'abc'.upper()	'ABC'
<code>str.lower()</code>	Converts to lowercase	'ABC'.lower()	'abc'
<code>str.capitalize()</code>	Capitalizes first character	'hello'.capitalize()	'Hello'
<code>str.title()</code>	Capitalizes each word	'hello world'.title()	'Hello World'
<code>str.swapcase()</code>		'PyThOn'.swapcase()	'pYTHON'

46:38

THE ML HUB

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

String Methods

Searching & Finding Methods

't t t t t t'.find("tt")
① ② ③

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

54:33 / 1:58:19

THE ML HUB

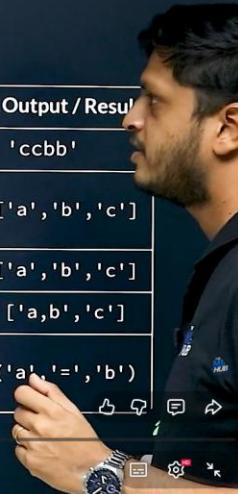
String Methods

Replacing & Splitting

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

▶ ⏪ ⏩ 🔍

54:33 / 1:58:19



String Methods

Content Testing Methods

Method	Returns True if	Example	Output / Result
<code>str.isalpha()</code>	All characters are alphabetic	'abc'.isalpha()	True
<code>str.isdigit()</code>	All characters are digits	'123'.isdigit()	True
<code>str.isalnum()</code>	All alphanumeric	'abc123'.isalnum()	True
<code>str.islower()</code>	All lower case	'abc'.islower()	True
<code>str.isupper()</code>	All upper case	'ABC'.isupper()	True
<code>str.isspace()</code>	All whitespace	' '.isspace()	True

Escape Characters

What Are Escape Characters in Python?

- Special sequences of characters used within strings to represent invisible or reserved characters
- They cannot be typed directly
- They start with a backslash and are interpreted in a special way

Examples:

- \n (Newline) ✓
- \' (Single quote)
- \\" (Backslash)

S = 'abc\'def'

abc'def

abc'def ✓

Escape Characters

What Are Escape Characters in Python?

- Special sequences of characters used within strings to represent invisible or reserved characters
- They cannot be typed directly
- They start with a backslash and are interpreted 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 Escape Characters in Python?

- Special characters used to represent invisible or reserved characters within strings
- They cannot be printed directly
- They start with a backslash \ to be interpreted in a special way

Example:

```
\n(1) abc\nef ✓  
\'(2) abc\nef ✓  
' (Double quote) ✓
```

abc\nef
abc"def ✓

THE ML HUB

`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 string.

```
s = "Gate2025"
```

```
print(len(s)) # Output: 8
```

`len(s) → 8`

`type(s) → class`

```
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 string after executing:

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

"gate" + ("2025" * 2)
⇒ "gate" + "20252025"
⇒ "gate20252025"

