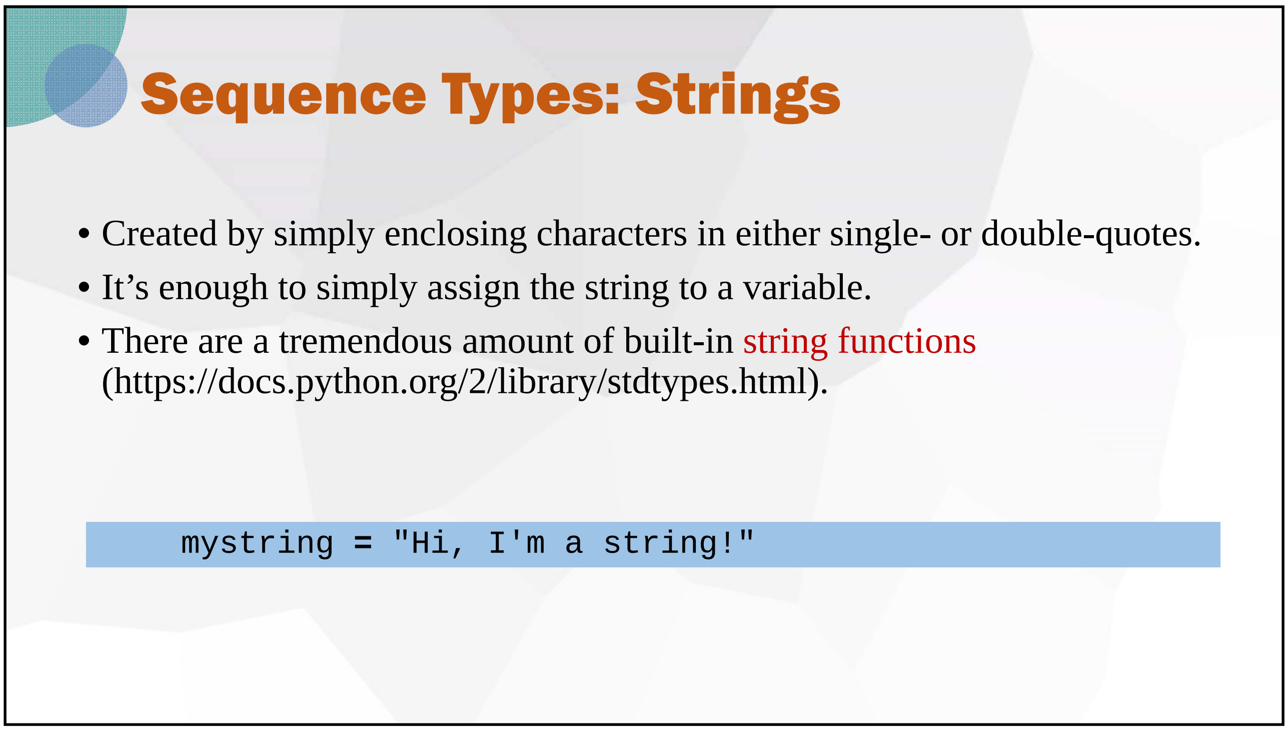
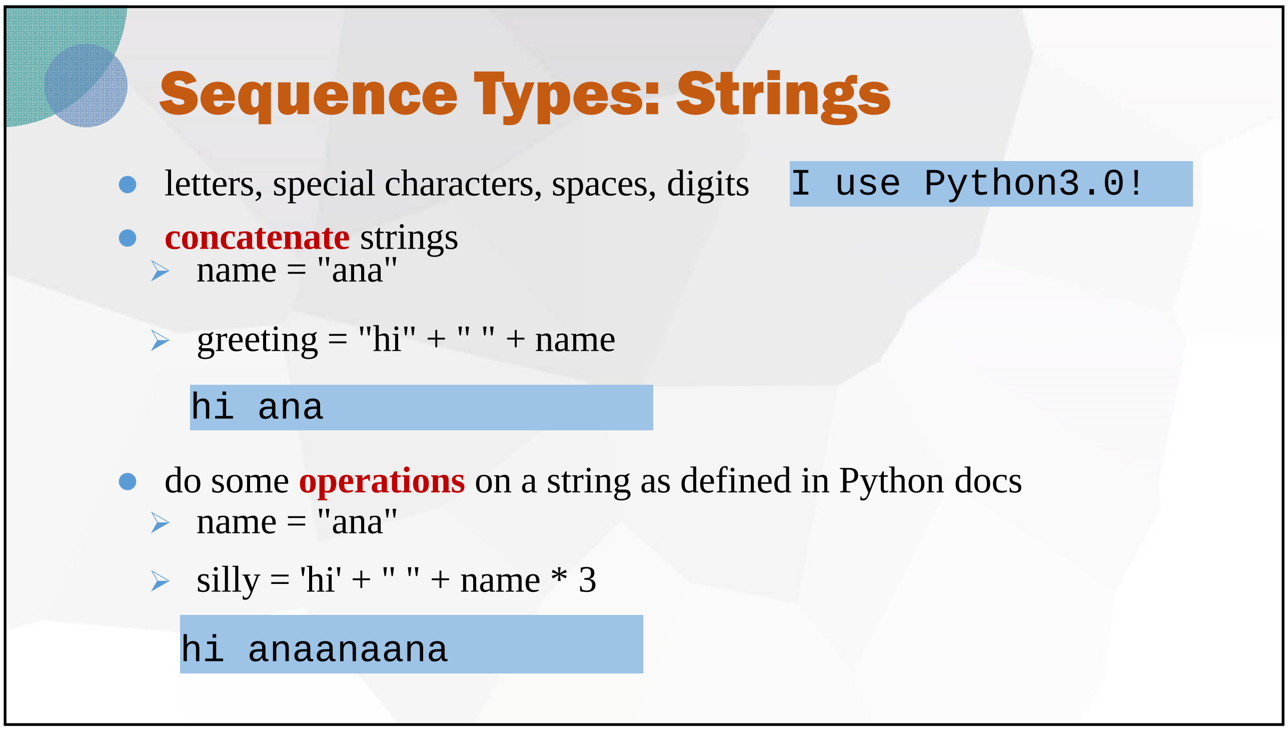


Sequence Types: Strings

- Created by simply enclosing characters in either single- or double-quotes.
- It’s enough to simply assign the string to a variable.
- There are a tremendous amount of built-in string functions (https://docs.python.org/2/library/stdtypes.html).

```
mystring = "Hi, I'm a string!"
```



Sequence Types: Strings

- letters, special characters, spaces, digits

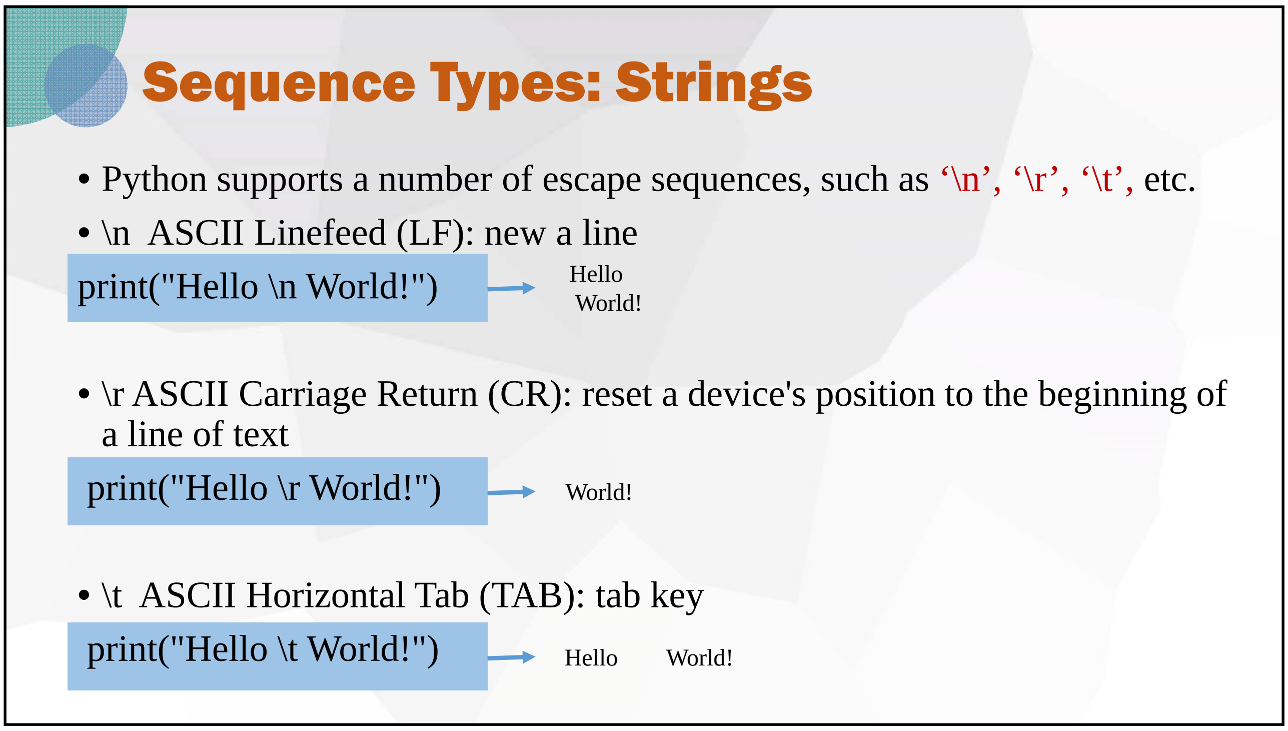
```
I use Python3.0!
```

- **concatenate** strings
  - name = "ana"
  - greeting = "hi" + " " + name

```
hi ana
```

- do some **operations** on a string as defined in Python docs
  - name = "ana"
  - silly = 'hi' + " " + name * 3

```
hi anaanaana
```



Sequence Types: Strings

- Python supports a number of escape sequences, such as ‘\n’, ‘\r’, ‘\t’, etc.
- \n ASCII Linefeed (LF): new a line

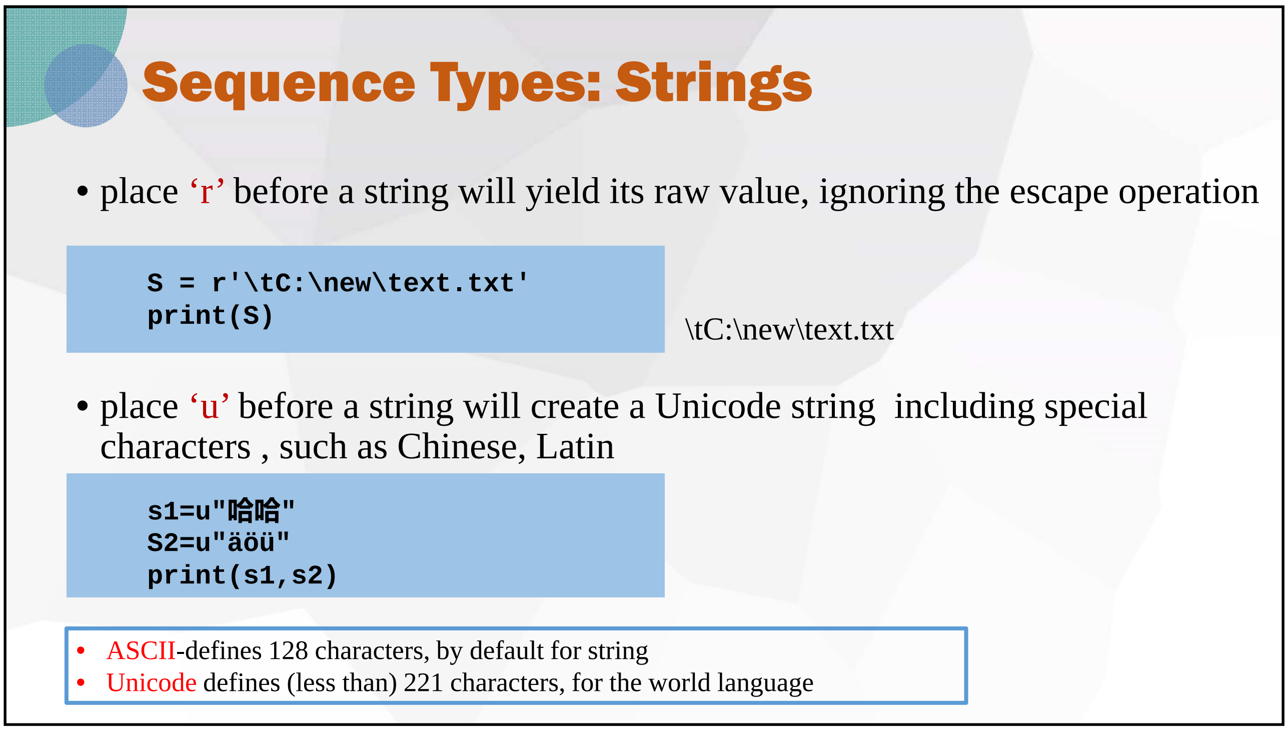
```
print("Hello \n World!")
```
→
```
Hello
 World!
```

- \r ASCII Carriage Return (CR): reset a device's position to the beginning of a line of text

```
print("Hello \r World!")
```
→
```
World!
```

- \t ASCII Horizontal Tab (TAB): tab key

```
print("Hello \t World!")
```
→
```
Hello       World!
```



Sequence Types: Strings

- place ‘r’ before a string will yield its raw value, ignoring the escape operation

```
S = r'\tC:\new\text.txt'
print(S)
```

```
\tC:\new\text.txt
```

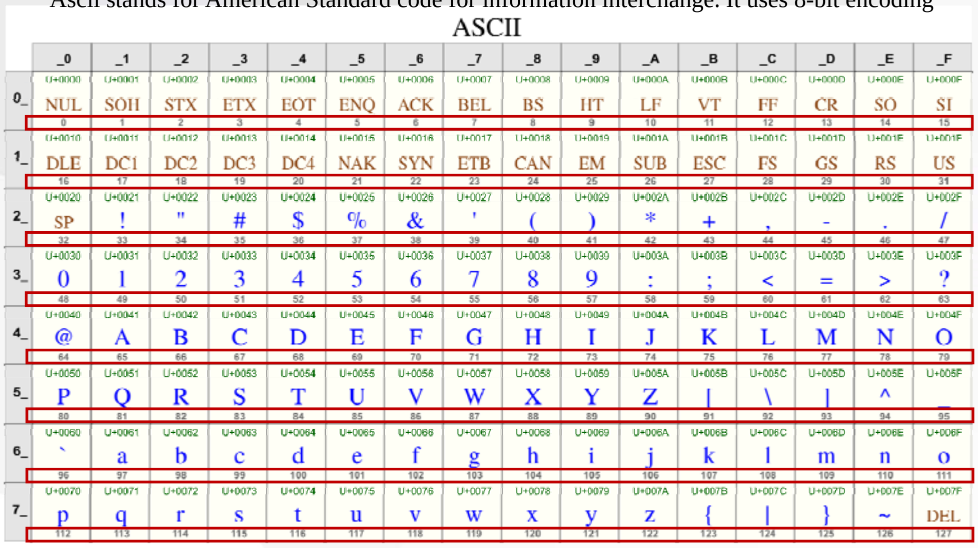
- place ‘u’ before a string will create a Unicode string including special characters , such as Chinese, Latin

```
s1=u"哈哈"
S2=u"äöü"
print(s1,s2)
```

- ASCII-defines 128 characters, by default for string
- Unicode defines (less than) 221 characters, for the world language

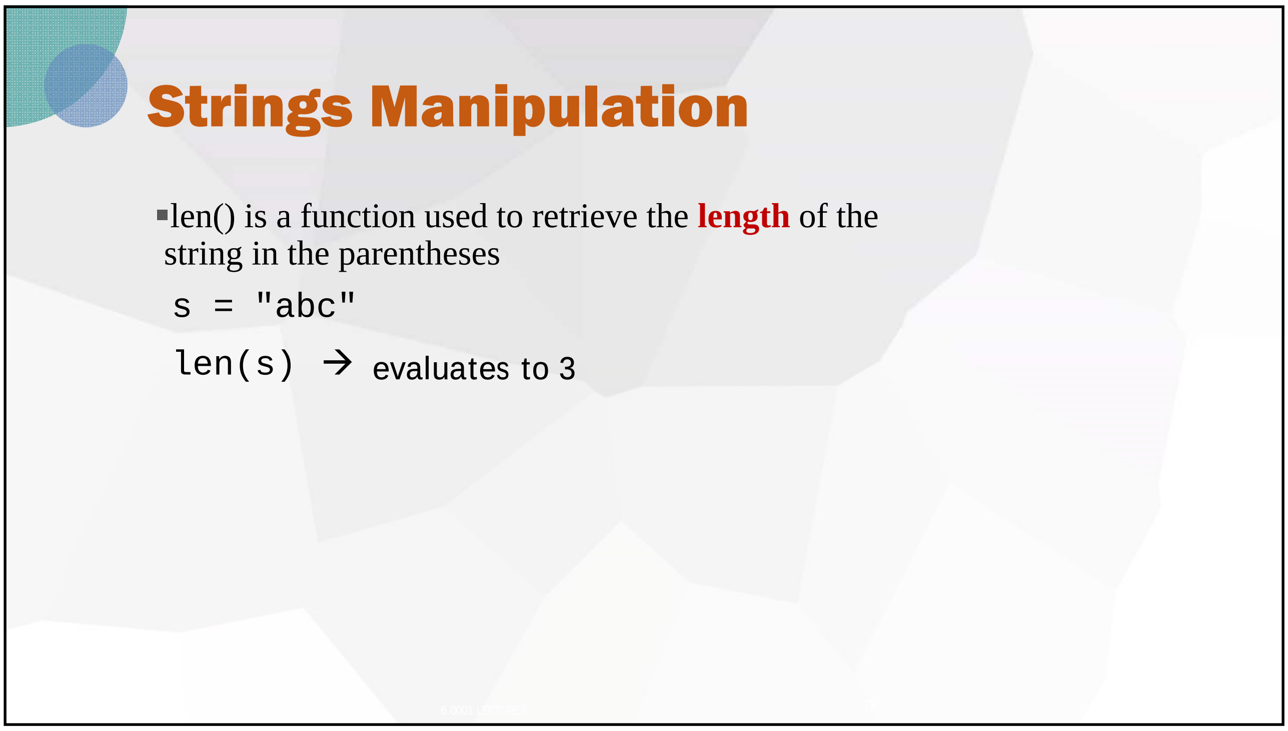
Sequence Types: Strings

Ascii stands for American Standard code for information interchange. It uses 8-bit encoding



ASCII

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	U+0000 NUL 0	U+0001 SOH 1	U+0002 STX 2	U+0003 ETX 3	U+0004 EOT 4	U+0005 ENQ 5	U+0006 ACK 6	U+0007 BEL 7	U+0008 BS 8	U+0009 HT 9	U+000A LF 10	U+000B VT 11	U+000C FF 12	U+000D CR 13	U+000E SO 14	U+000F SI 15
1_	U+0010 DLE 16	U+0011 DC1 17	U+0012 DC2 18	U+0013 DC3 19	U+0014 DC4 20	U+0015 NAK 21	U+0016 SYN 22	U+0017 ETB 23	U+0018 CAN 24	U+0019 EM 25	U+001A SUB 26	U+001B ESC 27	U+001C FS 28	U+001D GS 29	U+001E RS 30	U+001F US 31
2_	U+0020 SP 32	U+0021 ! 33	U+0022 " 34	U+0023 # 35	U+0024 $ 36	U+0025 % 37	U+0026 & 38	U+0027 ' 39	U+0028 (40	U+0029) 41	U+002A * 42	U+002B + 43	U+002C , 44	U+002D - 45	U+002E . 46	U+002F / 47
3_	U+0030 0 48	U+0031 1 49	U+0032 2 50	U+0033 3 51	U+0034 4 52	U+0035 5 53	U+0036 6 54	U+0037 7 55	U+0038 8 56	U+0039 9 57	U+003A : 58	U+003B ; 59	U+003C < 60	U+003D = 61	U+003E > 62	U+003F ? 63
4_	U+0040 @ 64	U+0041 A 65	U+0042 B 66	U+0043 C 67	U+0044 D 68	U+0045 E 69	U+0046 F 70	U+0047 G 71	U+0048 H 72	U+0049 I 73	U+004A J 74	U+004B K 75	U+004C L 76	U+004D M 77	U+004E N 78	U+004F O 79
5_	U+0050 P 80	U+0051 Q 81	U+0052 R 82	U+0053 S 83	U+0054 T 84	U+0055 U 85	U+0056 V 86	U+0057 W 87	U+0058 X 88	U+0059 Y 89	U+005A Z 90	U+005B [91	U+005C \ 92	U+005D] 93	U+005E ^ 94	U+005F _ 95
6_	U+0060 ` 96	U+0061 a 97	U+0062 b 98	U+0063 c 99	U+0064 d 100	U+0065 e 101	U+0066 f 102	U+0067 g 103	U+0068 h 104	U+0069 i 105	U+006A j 106	U+006B k 107	U+006C l 108	U+006D m 109	U+006E n 110	U+006F o 111
7_	U+0070 p 112	U+0071 q 113	U+0072 r 114	U+0073 s 115	U+0074 t 116	U+0075 u 117	U+0076 v 118	U+0077 w 119	U+0078 x 120	U+0079 y 121	U+007A z 122	U+007B { 123	U+007C | 124	U+007D } 125	U+007E ~ 126	U+007F DEL 127

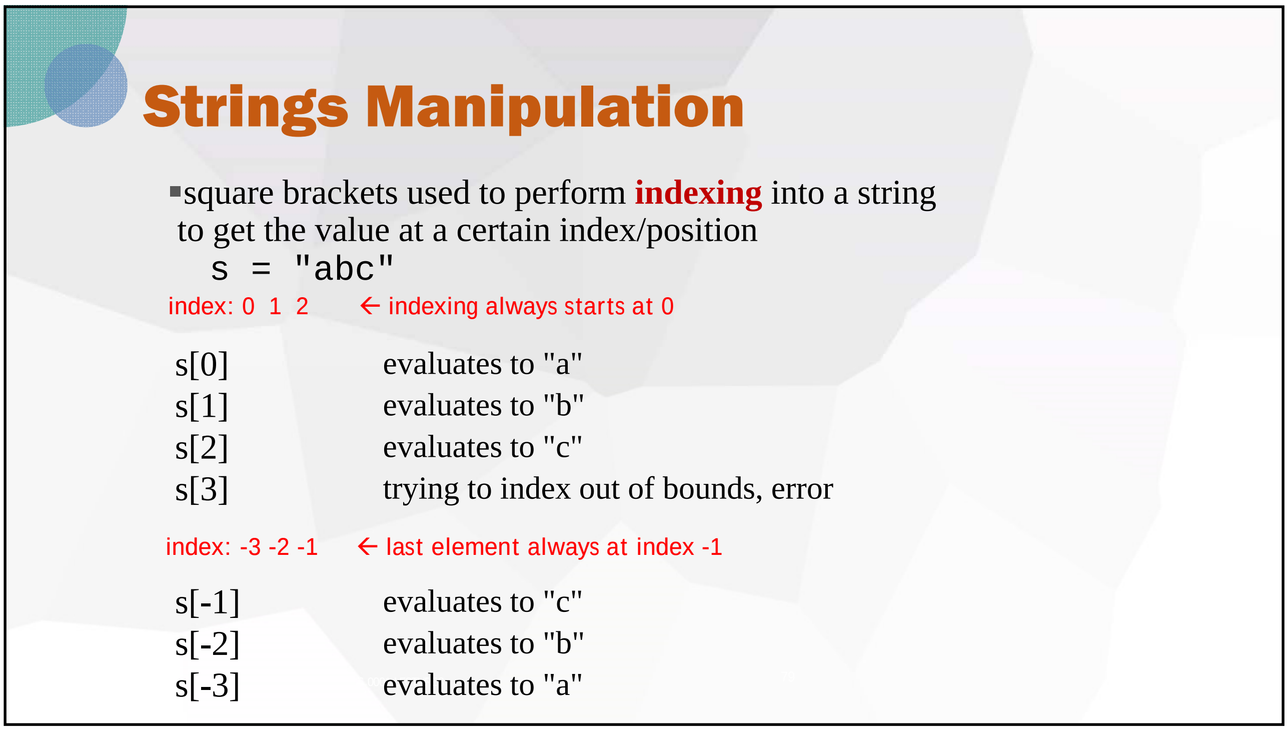


Strings Manipulation

- len() is a function used to retrieve the **length** of the string in the parentheses

```
s = "abc"
```

`len(s)` → evaluates to 3



Strings Manipulation

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

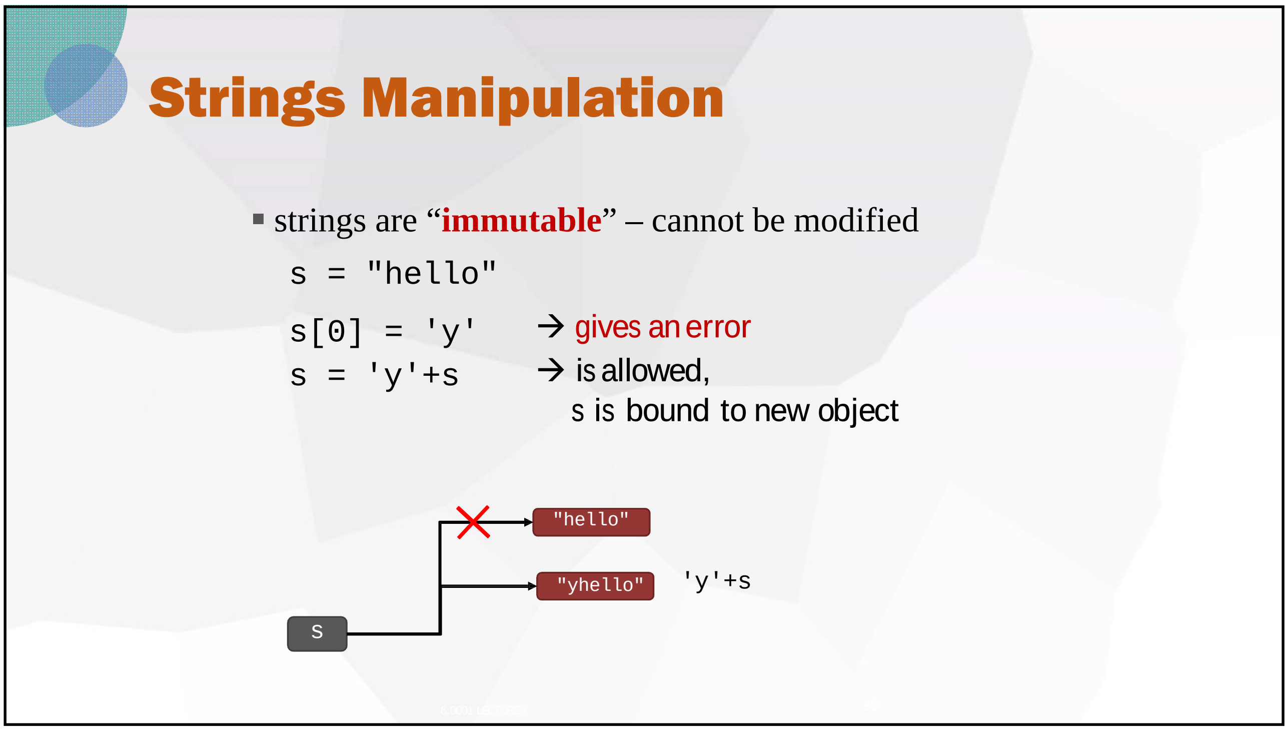
```
  s = "abc"
```

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

index: -3 -2 -1 ← last element always at index -1

s[-1]	evaluates to "c"
s[-2]	evaluates to "b"
s[-3]	evaluates to "a"

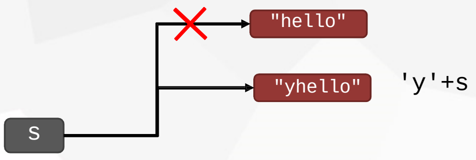


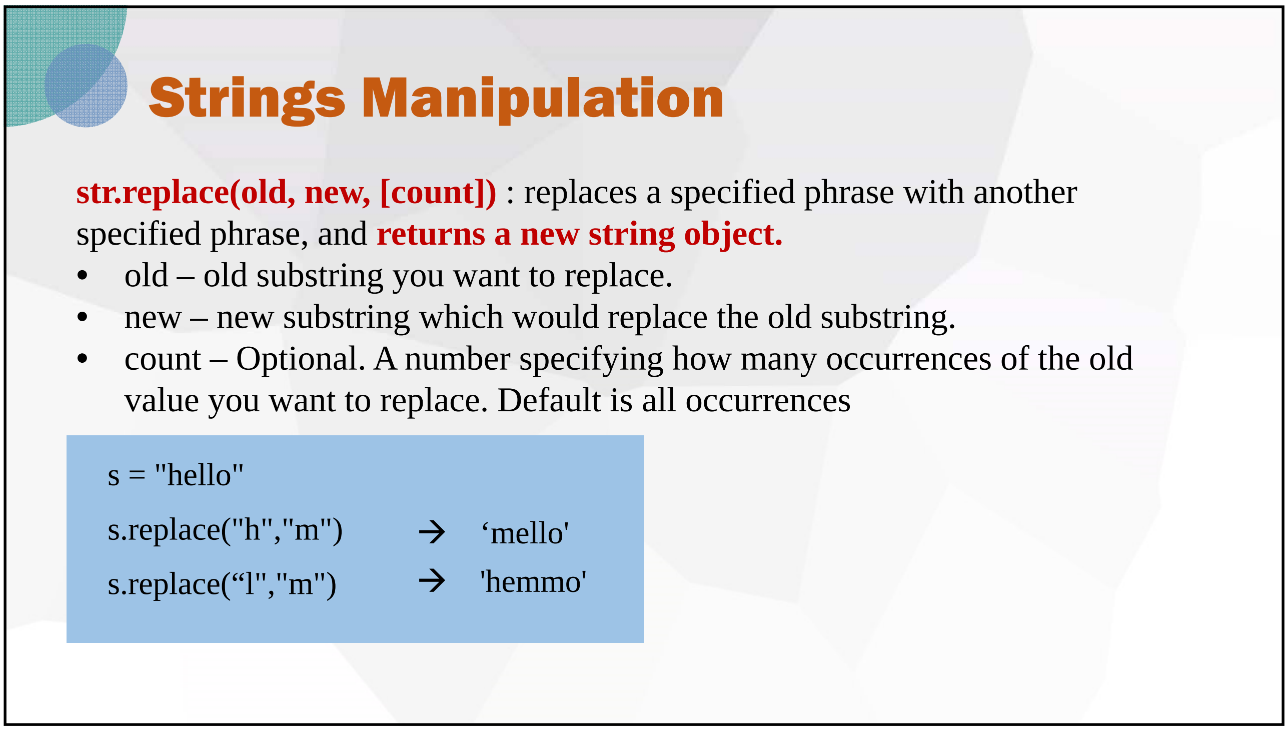
Strings Manipulation

- strings are “**immutable**” – cannot be modified

```
s = "hello"

s[0] = 'y'     → gives an error
s = 'y'+s      → is allowed,
                 s is bound to new object
```



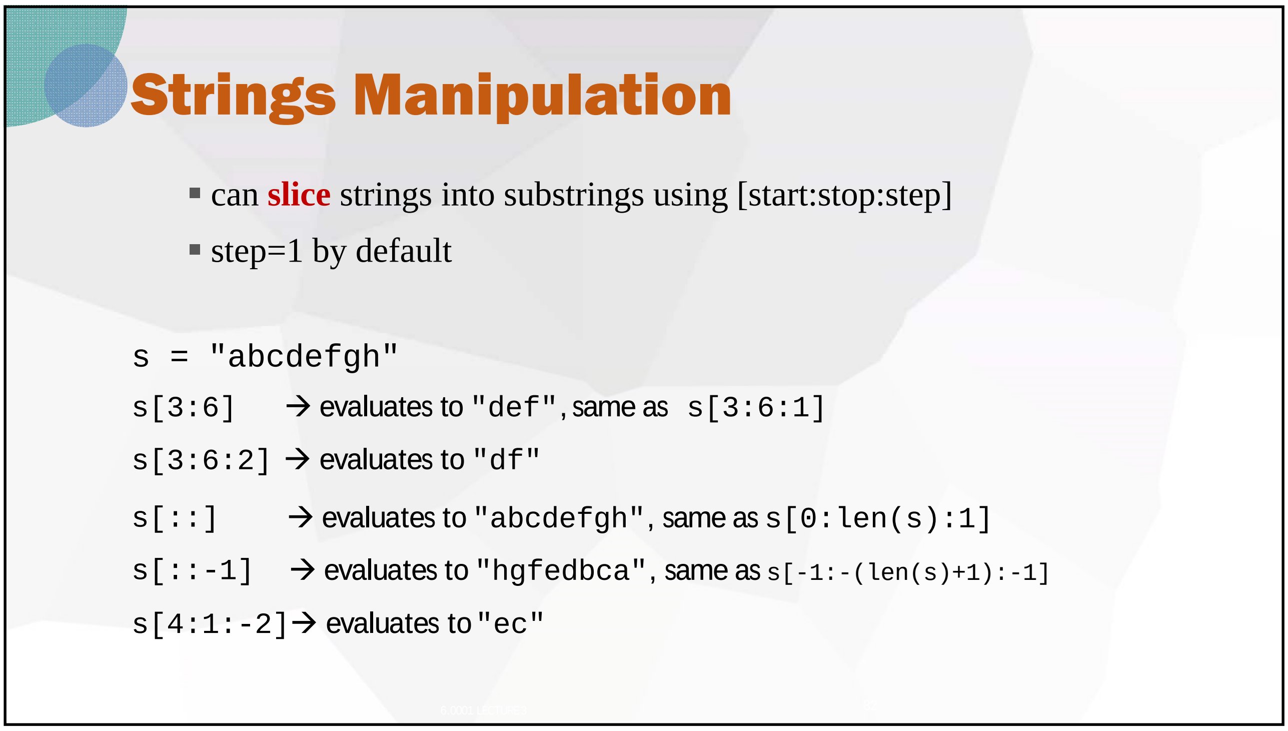


Strings Manipulation

str.replace(old, new, [count]) : replaces a specified phrase with another specified phrase, and **returns a new string object.**

- old – old substring you want to replace.
- new – new substring which would replace the old substring.
- count – Optional. A number specifying how many occurrences of the old value you want to replace. Default is all occurrences

```
s = "hello"
s.replace("h","m")      →   ‘mello'
s.replace(“l","m")      →   'hemmo'
```



Strings Manipulation

- can **slice** strings into substrings using [start:stop:step]
- step=1 by default

`s = "abcdefgh"`

`s[3:6]` → evaluates to `"def"`, same as `s[3:6:1]`

`s[3:6:2]` → evaluates to `"df"`

`s[::]` → evaluates to `"abcdefgh"`, same as `s[0:len(s):1]`

`s[::-1]` → evaluates to `"hgfedbca"`, same as `s[-1:-(len(s)+1):-1]`

`s[4:1:-2]`→ evaluates to `"ec"`

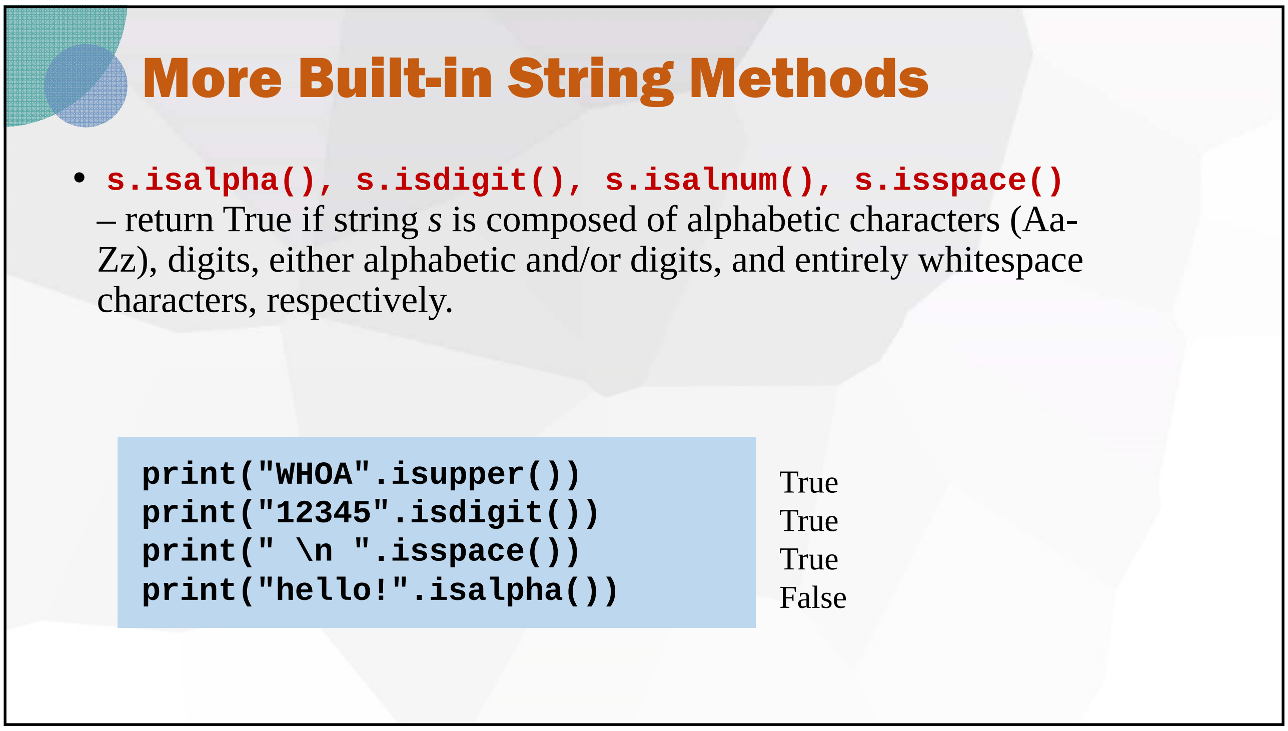
More Built-in String Methods

- Python includes a number of built-in string methods that are incredibly useful for string manipulation. Note that these methods return **the modified string value** since string is immutable.
- `s.upper()` and `s.lower()` converts all of the characters to uppercase or lowercase

```
s1 = "Python is so awesome."
print(s1.upper())
print(s1.lower())
```

```
PYTHON IS SO AWESOME.
python is so awesome.
```

- `s.islower(), s.isupper()` – return True if string *s* is all lowercase and all uppercase, respectively.

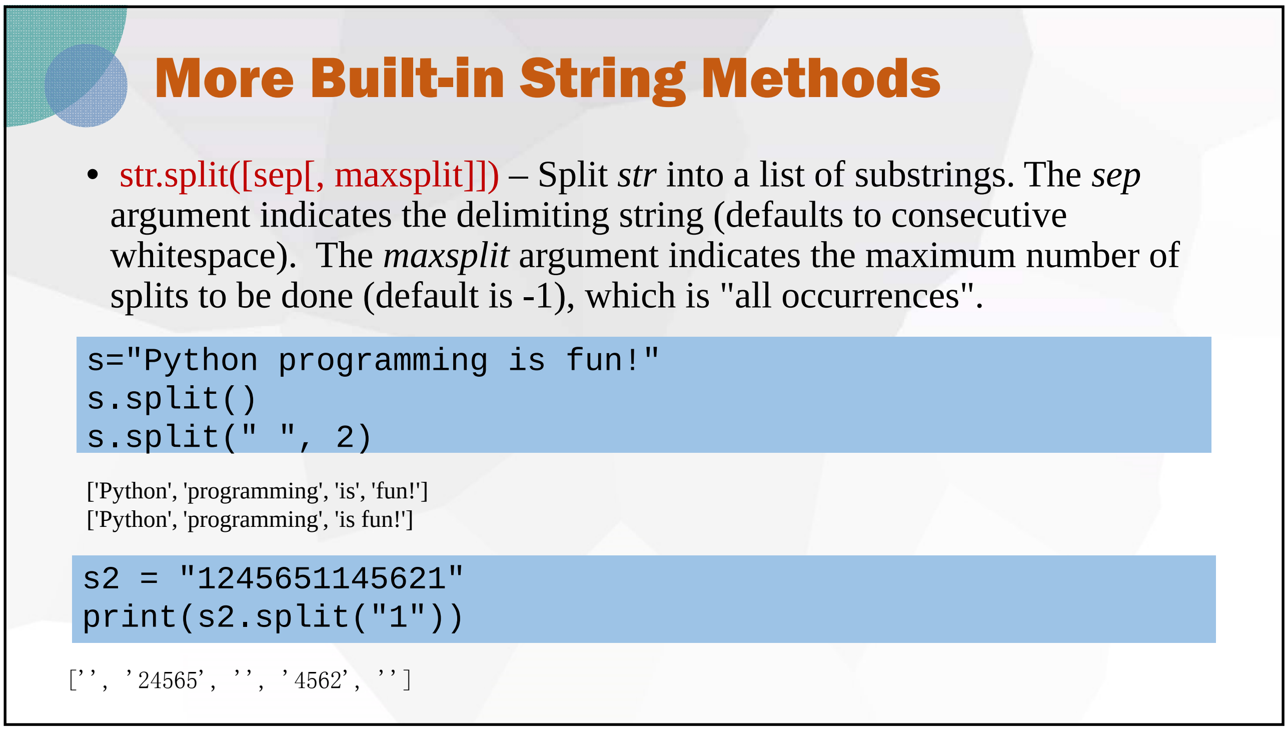


More Built-in String Methods

- `s.isalpha(), s.isdigit(), s.isalnum(), s.isspace()` – return True if string *s* is composed of alphabetic characters (Aa-Zz), digits, either alphabetic and/or digits, and entirely whitespace characters, respectively.

```
print("WHOA".isupper())
print("12345".isdigit())
print(" \n ".isspace())
print("hello!".isalpha())
```

```
True
True
True
False
```



More Built-in String Methods

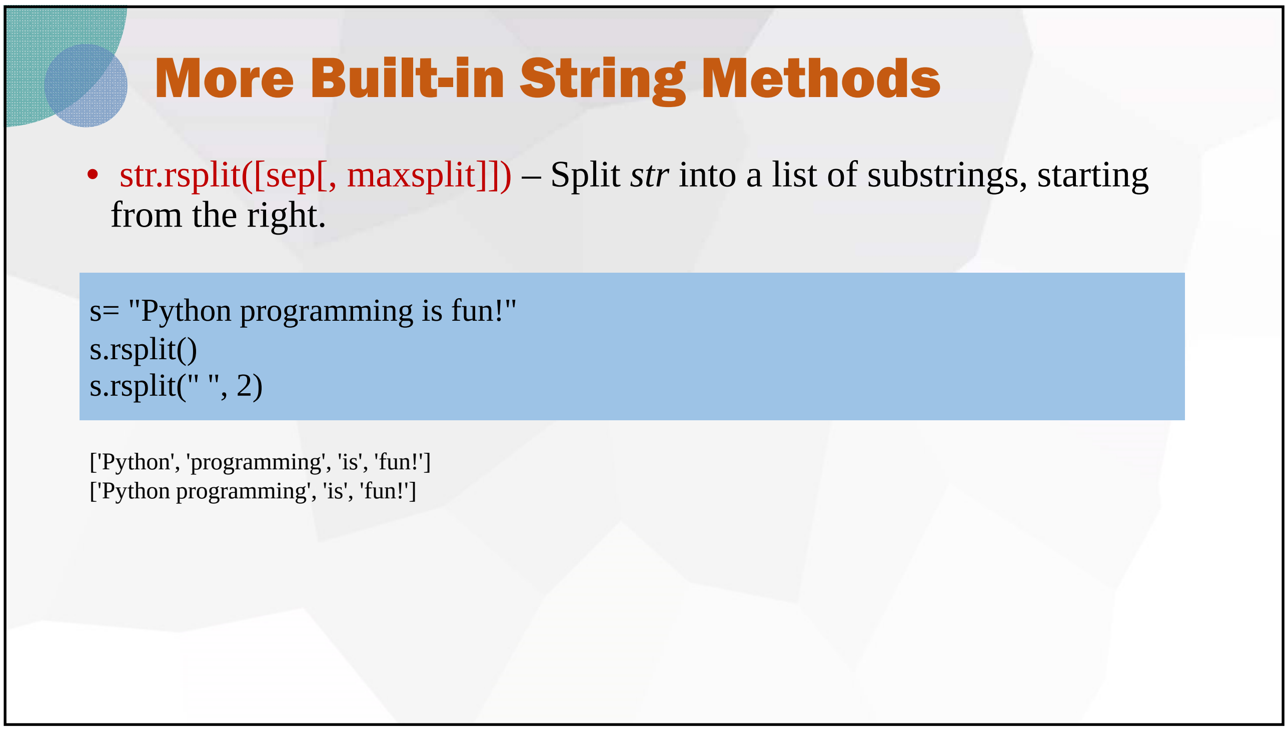
- str.split([sep[, maxsplit]]) – Split *str* into a list of substrings. The *sep* argument indicates the delimiting string (defaults to consecutive whitespace). The *maxsplit* argument indicates the maximum number of splits to be done (default is -1), which is "all occurrences".

```
s="Python programming is fun!"
s.split()
s.split(" ", 2)
```

```
['Python', 'programming', 'is', 'fun!']
['Python', 'programming', 'is fun!']
```

```
s2 = "1245651145621"
print(s2.split("1"))
```

```
[’’, ’24565’, ’’, ’4562’, ’’]
```

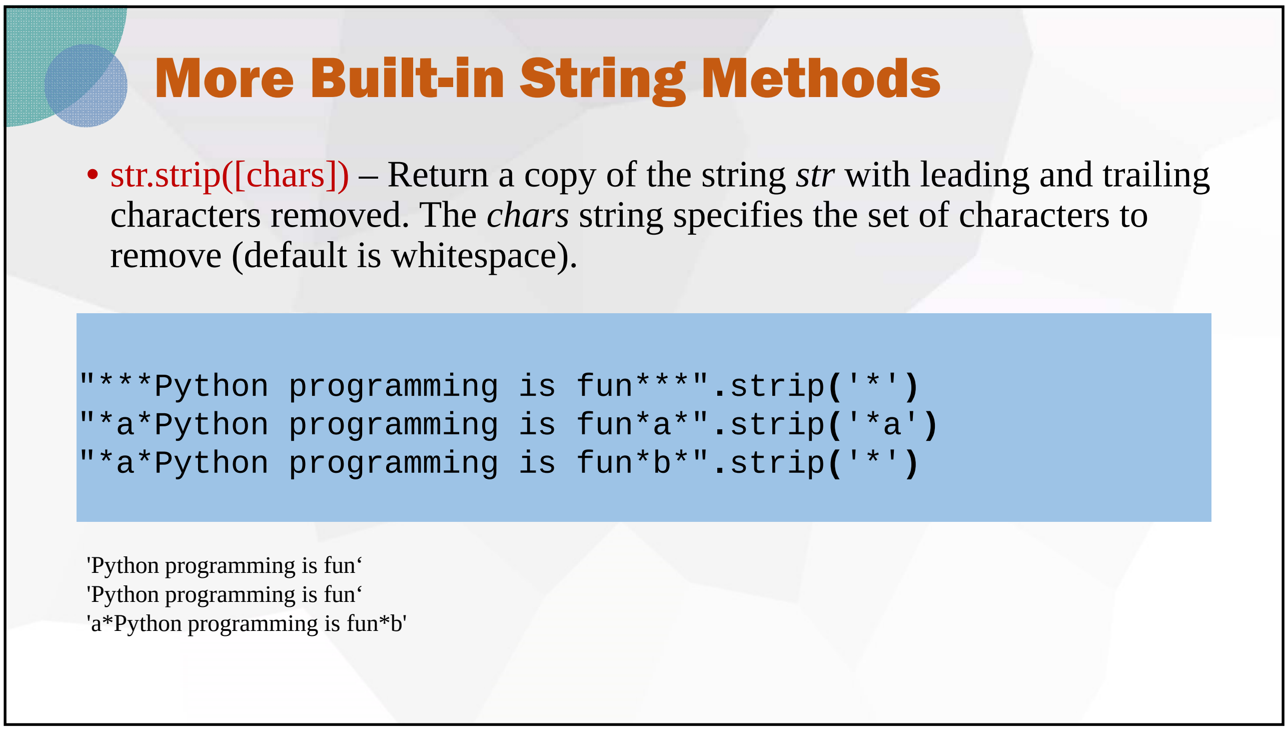


More Built-in String Methods

- str.rsplit([sep[, maxsplit]]) – Split *str* into a list of substrings, starting from the right.

```
s= "Python programming is fun!"
s.rsplit()
s.rsplit(" ", 2)
```

```
['Python', 'programming', 'is', 'fun!']
['Python programming', 'is', 'fun!']
```



More Built-in String Methods

- str.strip([chars]) – Return a copy of the string *str* with leading and trailing characters removed. The *chars* string specifies the set of characters to remove (default is whitespace).

```
"***Python programming is fun***".strip('*')
"*a*Python programming is fun*a*".strip('*a')
"*a*Python programming is fun*b*".strip('*')
```

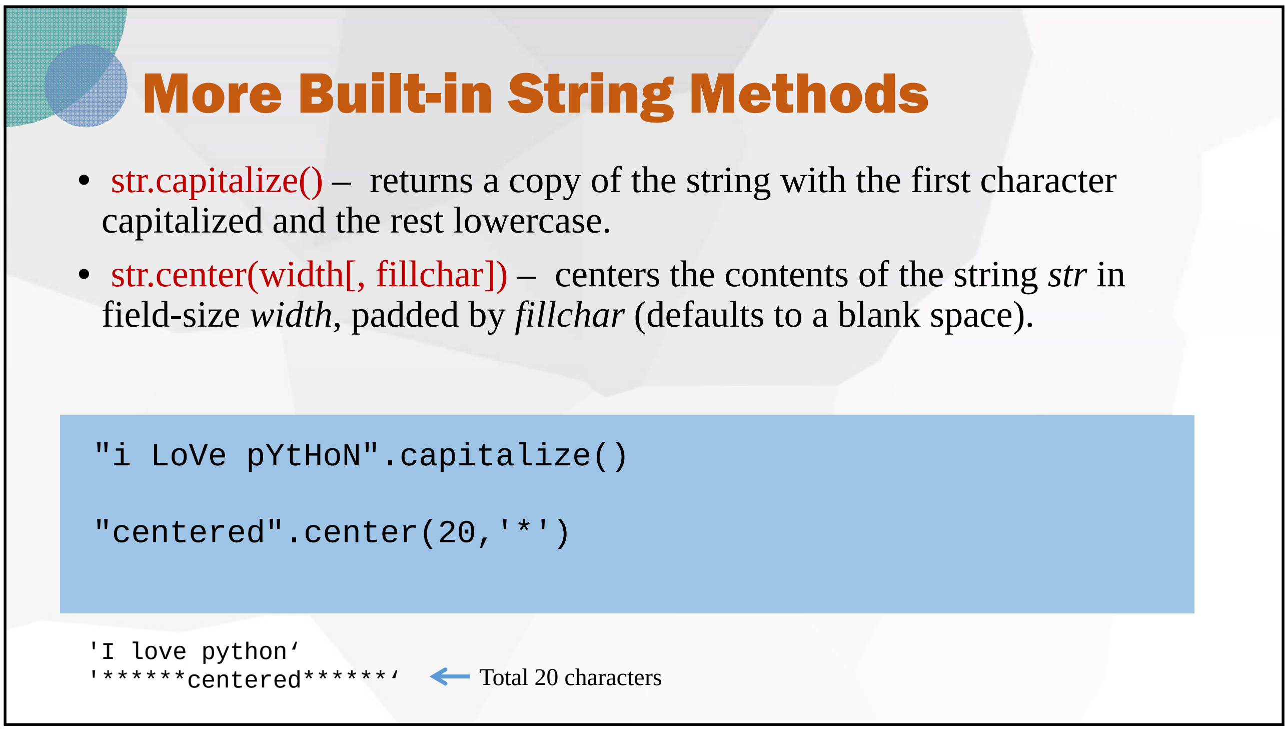
```
'Python programming is fun‘
'Python programming is fun‘
'a*Python programming is fun*b'
```

More Built-in String Methods

- str.rstrip([chars]) – Return a copy of the string *str* with only trailing characters removed.

```
"***Python programming is fun***".rstrip('*')
"*a*Python programming is fun*a*".rstrip('*a')
"*a*Python programming is fun*b*".rstrip('*')
```

```
'***Python programming is fun'
'*a*Python programming is fun‘
'*a*Python programming is fun*b'
```



More Built-in String Methods

- str.capitalize() – returns a copy of the string with the first character capitalized and the rest lowercase.
- str.center(width[, fillchar]) – centers the contents of the string *str* in field-size *width*, padded by *fillchar* (defaults to a blank space).

```
"i LoVe pYtHoN".capitalize()

"centered".center(20,'*')
```

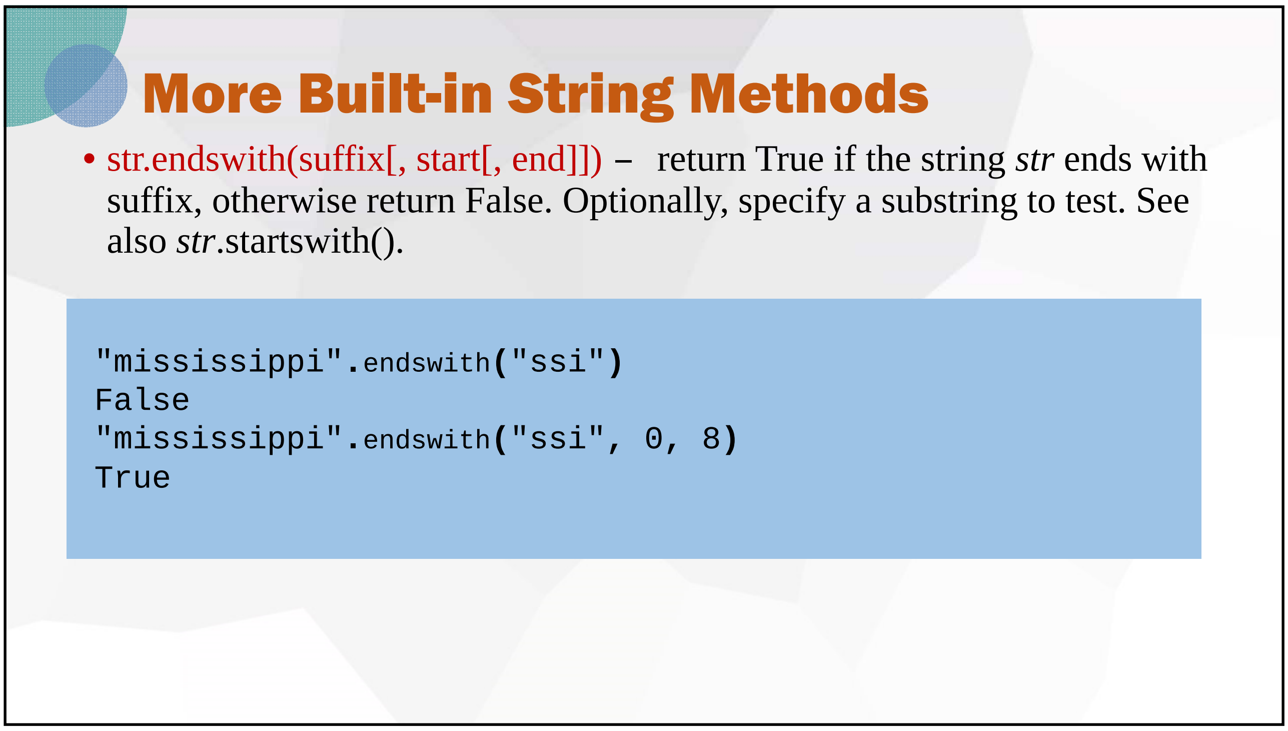
```
'I love python‘
'******centered******‘
```

Total 20 characters

More Built-in String Methods

- str.count(sub[, start[, end]]) – return the number of non-overlapping occurrences of substring *sub* in the range *[start, end]*.

```
"mississippi".count("iss")
2
"mississippi".count("iss", 4, -1)
1
```



More Built-in String Methods

- str.endswith(suffix[, start[, end]]) – return True if the string *str* ends with suffix, otherwise return False. Optionally, specify a substring to test. See also *str*.startswith().

```
"mississippi".endswith("ssi")
False
"mississippi".endswith("ssi", 0, 8)
True
```