# CMSC201
# Computer Science I for Majors

## Lecture 09 – Strings

# Last Class We Covered

- Lists and what they are used for
  - Getting the length of a list
  - Operations like **`append()`** and **`remove()`**
  - Iterating over a list using a **`while`** loop
  - Indexing
- Membership "**`in`**" operator
- Methods vs Functions

# Any Questions from Last Time?

# Today's Objectives

- To better understand the string data type
  - Learn how they are represented
  - Learn about and use some of their built-in methods
    - Slicing and concatenation
    - Escape sequences
    - lower() and upper()
    - strip() and whitespace
    - split() and join()

# Strings

# The String Data Type

- Text is represented in programs by the string data type
- A ***string*** is a sequence of characters enclosed within double quotes (") or single quotes (')
  - Sometimes called quotation marks or apostrophes

# Getting Strings as Input

- Using **`input()`** automatically gets a string

```
>>> firstName = input("Please enter your name: ")
Please enter your name: Shakira
>>> type(firstName)
<class 'str'>
>>> print(firstName, firstName)
Shakira Shakira
```

# Accessing Individual Characters

- We can access the individual characters in a string through ***indexing***
  - Characters are the letters, numbers, spaces, and symbols that make up a string

- The characters in a string are numbered starting from the left, beginning with 0
  - Just like in lists!

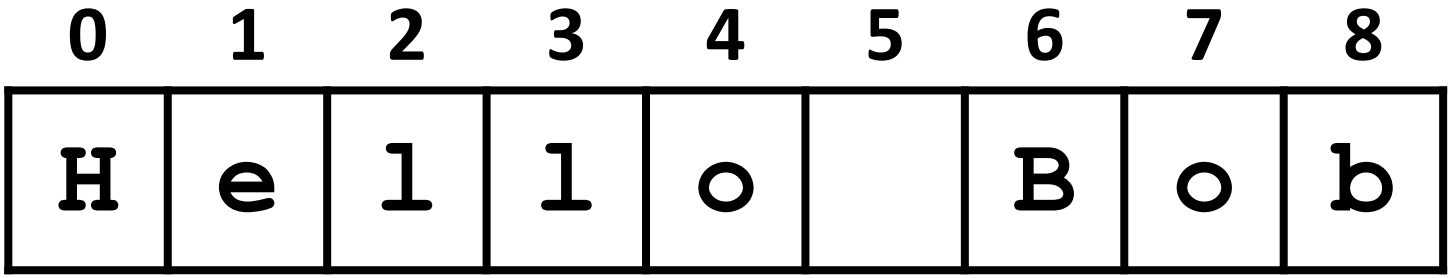
# Syntax of Accessing Characters

- The general form is

```
strName[expression]
```

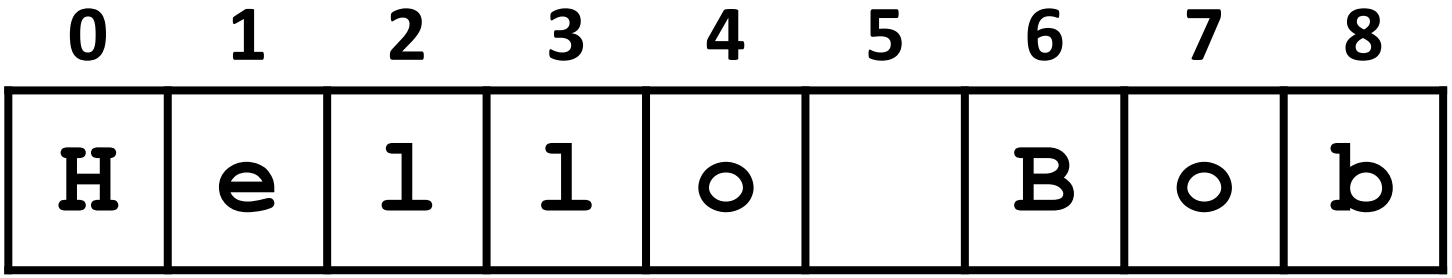
- Where **`strName`** is the name of the string variable and **`expression`** determines which character is selected from the string

# Example String




```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print(greet[0], greet[2], greet[4])
H l o
>>> x = 8
>>> print(greet[x - 2])
B
```

# Example String




- In a string of `n` characters, the last character is at position `n-1` since we start counting with 0

- So how can we access the last letter, regardless of the string's length?

```
greet[ len(greet) - 1 ]
```

# Changing String Case

- Python has many, many ways to interact with strings, and we will cover them in detail soon
- For now, here are two very useful methods:
  - `s.lower()` – copy of `s` in all lowercase letters
  - `s.upper()` – copy of `s` in all uppercase letters
- Why would we need to use these?
  - Remember, Python is case-sensitive!

# Concatenation

# Forming New Strings - Concatenation

- We can put two or more strings together to form a longer string

- ***Concatenation*** "glues" two strings together

```
>>> "Peanut Butter" + "Jelly"
'Peanut ButterJelly'
>>> "Peanut Butter" + " & " + "Jelly"
'Peanut Butter & Jelly'
```

# Rules of Concatenation

- Concatenation does not automatically include spaces between the strings

```
>>> "Smash" + "together"
'Smashtogether'
```

- Concatenation can only be done with strings!
  - So how would we concatenate an integer?

```
>>> "CMSC " + str(201)
'CMSC 201'
```

# Common Use for Concatenation

- **`input()`** only accepts a single string
  - Can't use commas like we do with **`print()`**

- In order to create a single string for **`input()`**, you must use concatenation

```
classNum = 201
grade = input("Grade in " + str(classNum) + "? ")
```

# Sentinels and Concatenation

- To take full advantage of sentinel constants, use them in the input prompts as well

- Instead of:

```
name = input("Name, X to quit: ")
```

- Concatenate to include the sentinel constant

```
name = input("Name, " + EXIT + " to quit: ")
```

# Sentinels, `input()`, and Concatenation

- We can even get really lazy, and create the message string before using it in `input()`

```
SENTINEL = -1

def main():
    msg = "Enter a grade, or '" + str(SENTINEL) + "' to quit: "
    grade = int(input(msg))

    while grade != SENTINEL:
        print("Congrats on getting a", grade, "in the class!")
        grade = int(input(msg))

main()
```

don't forget to cast to string if needed

# Substrings and Slicing

# Substrings

- Indexing only returns a single character from the entire string

- We can access a ***substring*** using a process called ***slicing***

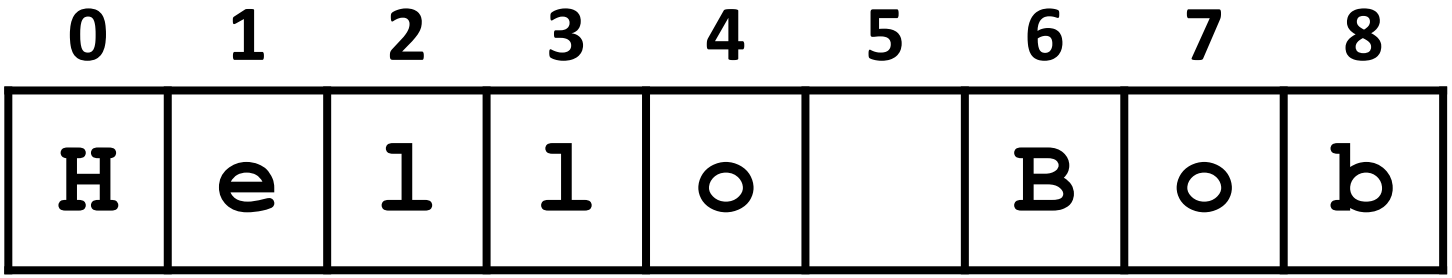
# Slicing Syntax

- The general form is

```
strName[start:end]
```

- **`start`** and **`end`** must evaluate to integers
  - The substring begins at index **`start`**
  - The substring ends **before** index **`end`**
    - The letter at index **`end`** is not included

# Slicing Examples




```
>>> greet[0:2]
'He'
>>> greet[7:9]
'ob'
>>> greet[:5]
'Hello'
>>> greet[1:]
'ello Bob'
>>> greet[:]
'Hello Bob'
```

# Specifics of Slicing

- If **`start`** or **`end`** are missing, then the start or end of the string is used instead

- The index of **`end`** must come after the index of **`start`**
  - What would the substring **`greet[1:1]`** be?

    ```
    ''
    ```

  - An empty string!

# String Operations in Python

| Operator | Meaning |
|---|---|
| `+` | Concatenation |
| `STRING[#]` | Indexing |
| `STRING[#:#]` | Slicing |
| `len(STRING)` | Length |

- All of this also applies to lists!
  - Two lists can be concatenated together
  - A sublist can be sliced from another list

# Escape Sequences

# Special Characters

- Just like Python has special keywords…
  - **`and`**, **`while`**, **`True`**, etc.
- It also has special characters
  - Single quote (`'`), double quote (`"`), etc.

- How can we print out a `"` as part of a string?

```
print("And I shouted "hey!" at him.")
```

  - What's going to happen here?
  - **`SyntaxError: EOL while scanning string literal`**

# Backslash: Escape Sequences

- The backslash character (\) is used to "***escape***" a special character in Python
  - Tells Python not to treat it as special

- The backslash character goes in front of the character we want to "escape"

```
>>> print("And I shouted \"hey!\"")
And I shouted "hey!"
```

# Common Escape Sequences

| Escape Sequence | Purpose |
|---|---|
| Escaping special characters | |
| `\'` | Print a single quote |
| `\"` | Print a double quote |
| `\\` | Print a backslash |
| Inserting a special character | |
| `\t` | Print a tab |
| `\n` | Print a new line (“enter”) |

# Escape Sequences Example

```
special1 = "I\tlove tabs."
print(special1)
I       love tabs.
```

`\t` adds a tab

```
special2 = "It's time to\nsplit!"
print(special2)
It's time to
split!
```

`\n` adds a newline

```
special3 = "Keep \\ em \\ separated"
print(special3)
Keep \ em \ separated
```

`\\` adds a single backslash

# Escape Sequences Example

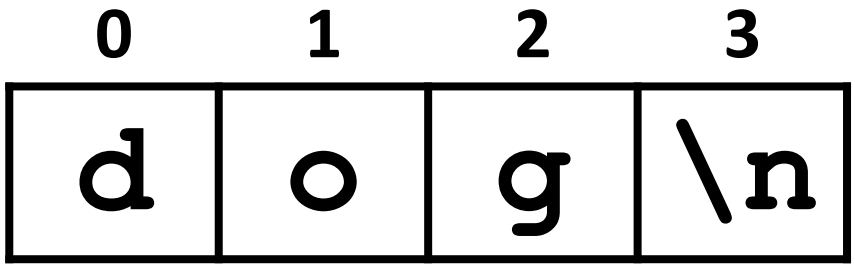
```
special1 = "I\tlove tabs."
print(special1)
I       love tabs.

special2 = "It's time to\nsplit!"
print(special2)
It's time to
split!

special3 = "Keep \\ em \\ separated"
print(special3)
Keep \ em \ separated
```

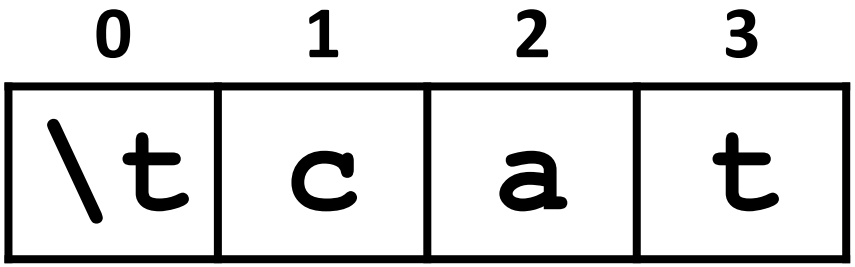
Note that there are no spaces around the escape sequences, but they work fine. What would happen if we added a space after `\t` or `\n` here?

# How Python Handles Escape Sequences

- Escape sequences look like two characters to us
- Python treats them as a single character

```
example1 = "dog\n"
example2 = "\tcat"
```



# The "end" of `print()`

- We've mentioned the use of `end=""` within a `print()` in a few of the homeworks
  - By default, `print()` uses `\n` as its ending

- We can use `end=` to change this

```
print("No newlines", end="")
print("More space please", end="\n\n")
print("Smile!", end=" :)\n")
```

  - Remember to put a `\n` in if you still want one!

# Whitespace

# Whitespace

- Whitespace is any “blank” character, that represents space between other characters
- For example: tabs, newlines, and spaces

```
"\t"   "\n"      " "
```

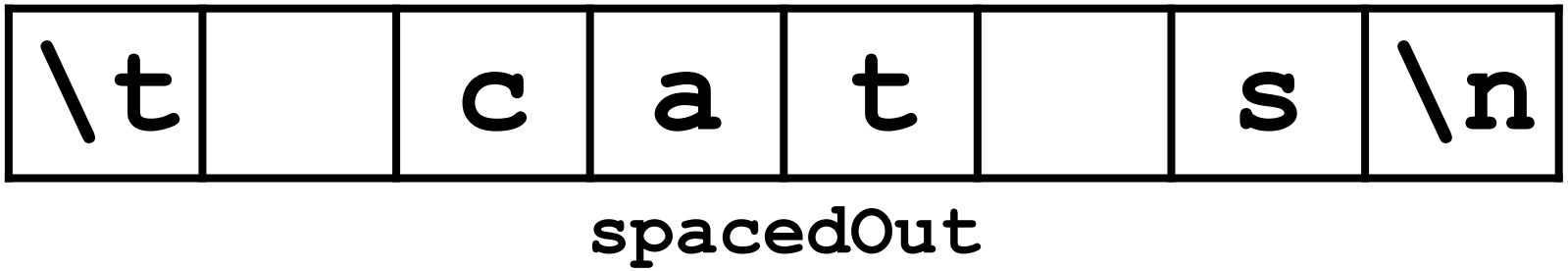
- Whitespace can cause similar strings to not be equivalent

```
>>> "dog" == " dog"
False
```

# Removing Whitespace

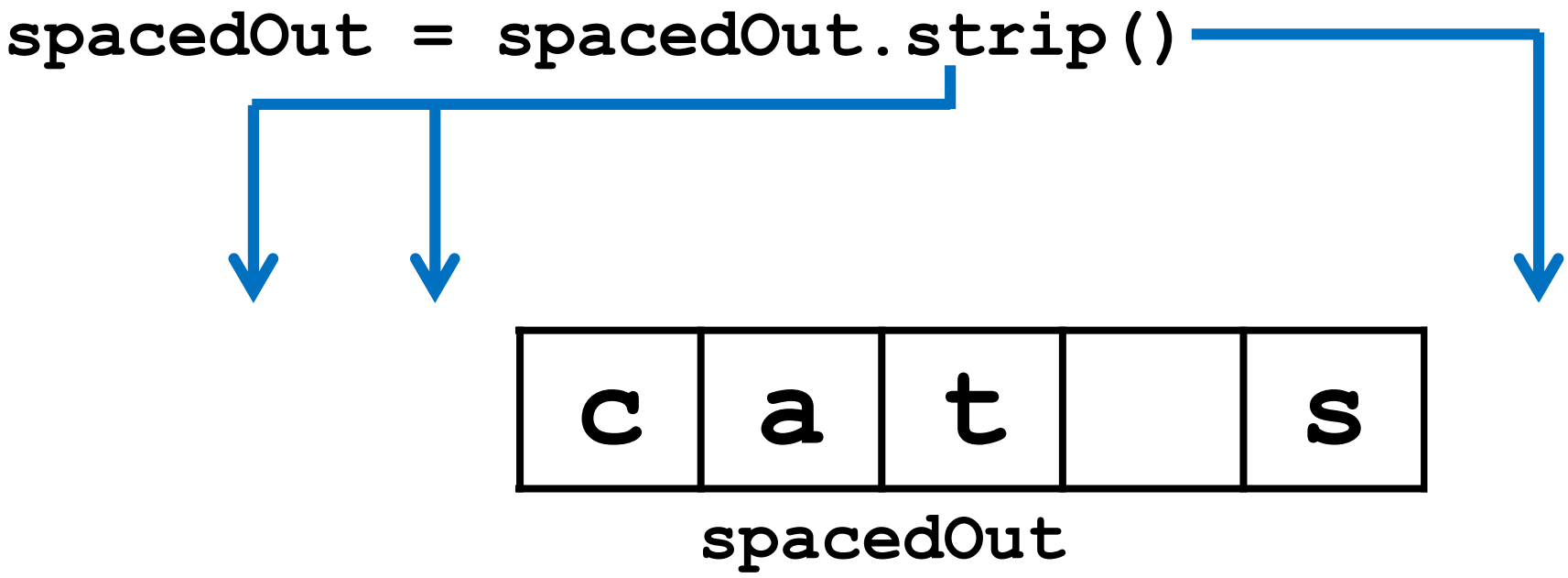
- To remove all whitespace from the start and end of a string, we can use a method called **`strip()`**

```
spacedOut = spacedOut.strip()
```



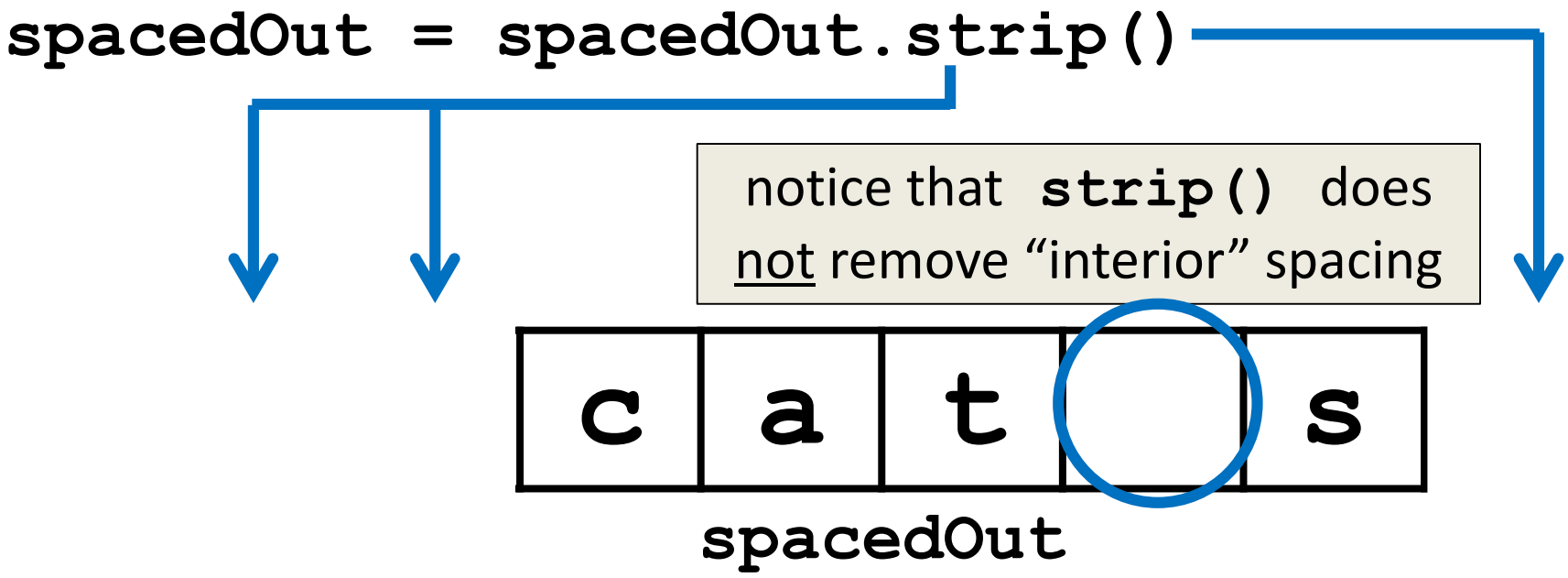
# Removing Whitespace

- To remove all whitespace from the start and end of a string, we can use a method called **`strip()`**



# Removing Whitespace

- To remove all whitespace from the start and end of a string, we can use a method called **`strip()`**



# String Splitting

# String Splitting

- We can also break a string into pieces
  - Stored as a list of strings

- The method is called **`split()`**, and it has two ways it can be used:
  - Break the string up by its whitespace
  - Break the string up by a specific character

# Splitting by Whitespace

- Calling **`split()`** with nothing inside the parentheses will split on all whitespace
  - Even the "interior" whitespace

```
>>> line = "hello world \n"
>>> line.split()
['hello', 'world']

>>> love = "\t\nI   love\t\t\nwhitespace\n  "
>>> love.split()
['I', 'love', 'whitespace']
```

# Splitting by Specific Character

- Calling **`split()`** with a string in it, we can remove a specific character (or more than one)

these character(s) that we want to remove are called the delimiter

```
>>> under = "once_twice_thrice"
>>> under.split("_")
['once', 'twice', 'thrice']

>>> double = "hello how ill are all of your llamas?"
>>> double.split("ll")
['he', 'o how i', ' are a', ' of your ', 'amas?']
```

# Splitting by Specific Character

- Calling **`split()`** with a string in it, we can remove a specific character (or more than one)

these character(s) that we want to remove are called the delimiter

```
>>> under = "once_twice_thrice"
>>> under.split("_")
['once', 'twice', 'thrice']
```

```
>>> double = "hello how ill are all of your llamas?"
>>> double.split("ll")
['he', 'o how i', ' are a', ' of your ', 'amas?']
```

notice that it didn't remove the whitespace

# Practice: Splitting

- Use **`split()`** to solve the following problems

- Split this string on its whitespace:

```
daft = "around \t the \nworld"
```

- Split this string on the double t's (**`tt`**):

```
adorable = "nutty otters making lattes"
```

# Practice: Splitting

- Use **`split()`** to solve the following problems

- Split this string on its whitespace:

```
daft = "around \t the \nworld"
daft.split()
```

- Split this string on the double t's (**`tt`**):

```
adorable = "nutty otters making lattes"
adorable.split("tt")
```

# Looping over Split Strings

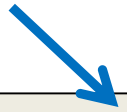
- Splitting a string creates a list of smaller strings
- Using a **`while`** loop and this list, we can iterate over each individual word (or token)

```
words = sentence.split()
index = 0
while index < len(words):
    print(words[index])
    index += 1
```

# Example: Looping over Split Strings

```
lyrics = "stars in their eyes"
lyricWords = lyrics.split()
index = 0
while index < len(lyricWords):
      print("*" + lyricWords[index] + "*")
      index += 1
```

```
*stars*
*in*
*their*
*eyes*
```



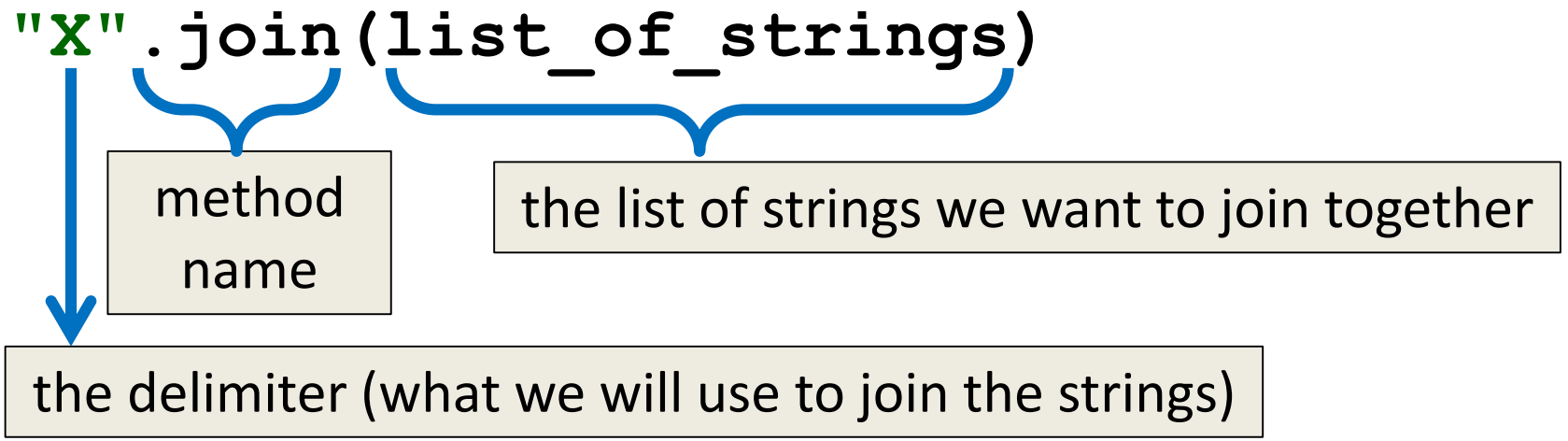
what does this line of code do?

append a "*" to the front and end of each list element, then print

# String Joining

# Joining Strings

- We can also join a list of strings back together!
  - The syntax looks different from **`split()`**
  - And it only works on a list of strings



# Example: Joining Strings

```
>>> names = ['Alice', 'Bob', 'Carl', 'Dana', 'Eve']
>>> "_".join(names)
'Alice_Bob_Carl_Dana_Eve'
```

- We can also use more than one character as our delimiter if we want

```
>>> " <3 ".join(names)
'Alice <3 Bob <3 Carl <3 Dana <3 Eve'
```

# `split()` vs `join()`

- The `split()` method
  - Takes in a single string
  - Creates a list of strings
  - Splits on given character(s), or on all whitespace

- The `join()` method
  - Takes in a list of strings
  - Returns a single string
  - Joins together with a user-chosen delimiter

# String and List Operations

- Many of the operations we've learned are possible to use on strings and on lists

| Operation | Strings | Lists |
|---|---|---|
| Concatenation + | ✓ | ✓ |
| Indexing [ ] | ✓ | ✓ |
| Slicing [ : ] | ✓ | ✓ |
| `.lower()` / `.upper()` | ✓ | ✗ |
| `.append()` / `.remove()` | ✗ | ✓ |
| `len()` | ✓ | ✓ |

# Daily Shortcut

- **`CTRL+Z`**
  - "Minimizes" the emacs window
- **`fg`**
  - Used in the terminal, and "maximizes" it again
- Useful when coding and testing
  - Save and minimize, run code, maximize it to edit
  - Keeps the kill ring, where you are in the file, etc.

# Announcements

- HW 3 is out on Blackboard now
- HW 4, Lab 6, review answer keys
  - Will be on Blackboard Saturday morning @ 10
- Midterm is in class, March 6th and 7th
  - That's next week, **Wednesday and Thursday**
    - **NOT** Monday and Tuesday!!!
  - Survey #1 will be released that week as well

# Image Sources

- Sewing thread (adapted from):
  - https://pixabay.com/p-936467
- Cheese slices:
  - http://pngimg.com/download/4276
- Space dog (adapted from):
  - https://commons.wikimedia.org/wiki/File:Space_dog_illustration.png