FILE HANDLING AND DICTIONARIES

Shannon Turner

Twitter: @svt827

Github: http://github.com/shannonturner

OBJECTIVE

- Review Lesson Two
- Learn how to read info from files
- Learn how and when to use dictionaries
- Using everything we've learned so far: strings, slicing, conditionals, lists, loops, file handling, dictionaries

LIGHTNING REVIEW

- Lists can hold multiple items at once
- Slicing allows us to view individual (or multiple) items in a list
- The **in** keyword allows us to check whether a given item appears in that list
- .append( ) adds one item to the end, .pop( ) removes one item from the end

HEAR ME CODE

LIGHTNING REVIEW

- Loops allow us to write code once but have it run multiple times
- For loops: for each item in this list, do something
- While loops: cousin of the conditional. "As long as I have enough bread, keep making sandwiches"

HEAR ME CODE

FILE HANDLING

- File handling lets Python read and write to files
  - Read from or write to a spreadsheet
  - Read from or write to a text file

FILE HANDLING: MOST COMMON SYNTAX

```
with open("states.txt", "r") as states_file:
    states = states_file.read()

print states
```

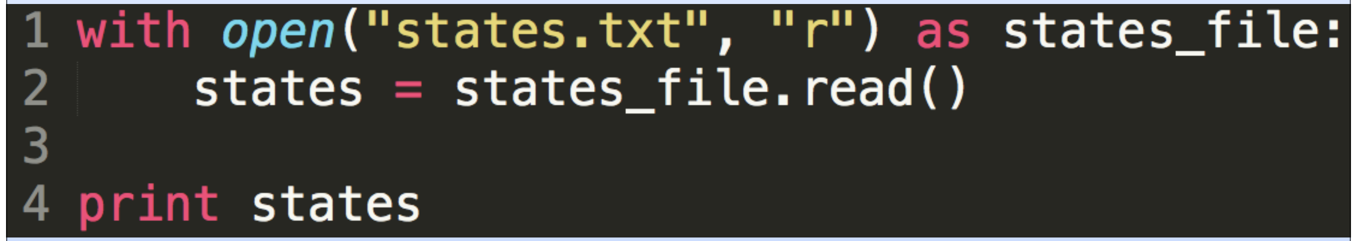
with keyword: tells Python we're going to do something **with** a file we're about to open.

When all commands within the indentation have been run, the file is closed automatically.

In the next few slides, we'll be exploring each part of the syntax individually.

We'll be going through these slides pretty quickly.

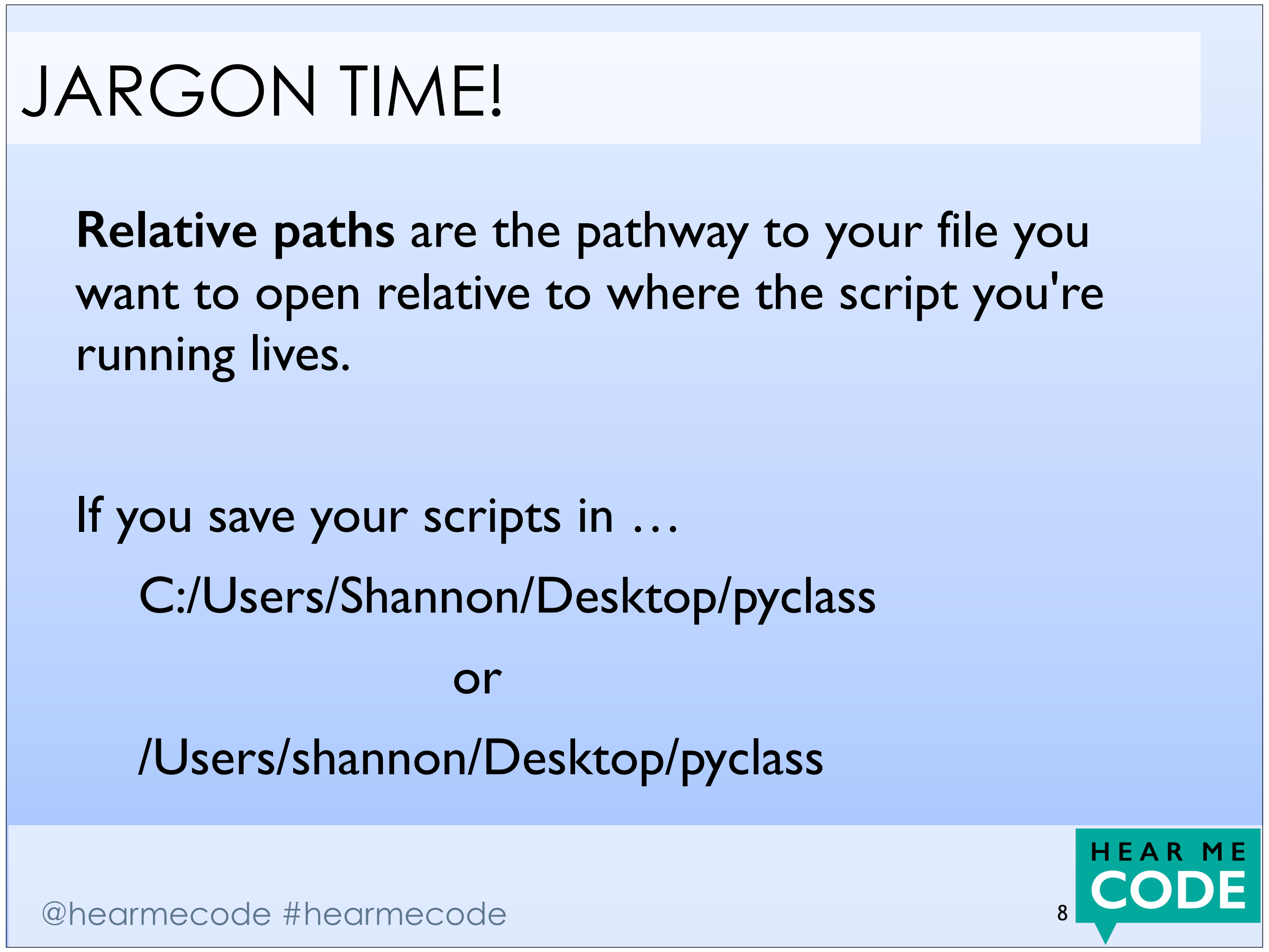
FILE HANDLING: MOST COMMON SYNTAX



```
with open("states.txt", "r") as states_file:
    states = states_file.read()

print states
```

open( ) built-in function, tells Python to open a file.

Argument 1: The file you want to open, using relative paths*



JARGON TIME!

Relative paths are the pathway to your file you want to open relative to where the script you're running lives.

If you save your scripts in …

C:/Users/Shannon/Desktop/pyclass

or

/Users/shannon/Desktop/pyclass

First example: windows

Second example: Mac / Linux

RELATIVE PATHS

If you save your scripts in …

C:/Users/Shannon/Desktop/pyclass

or

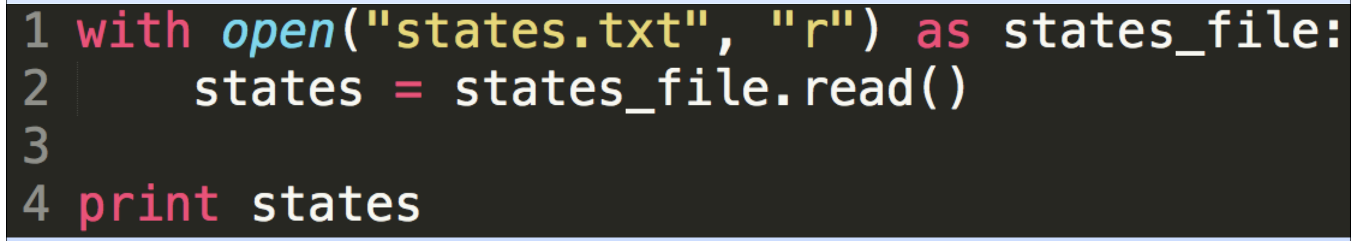
/Users/shannon/Desktop/pyclass

If your file and script are in the same folder, you can just tell Python the filename! (If not, where is the file you're opening **relative** to your script?)

Emphasize: but make it easy on yourself; put your files and scripts in the same folder.

This works similarly to URLs.

FILE HANDLING: MOST COMMON SYNTAX



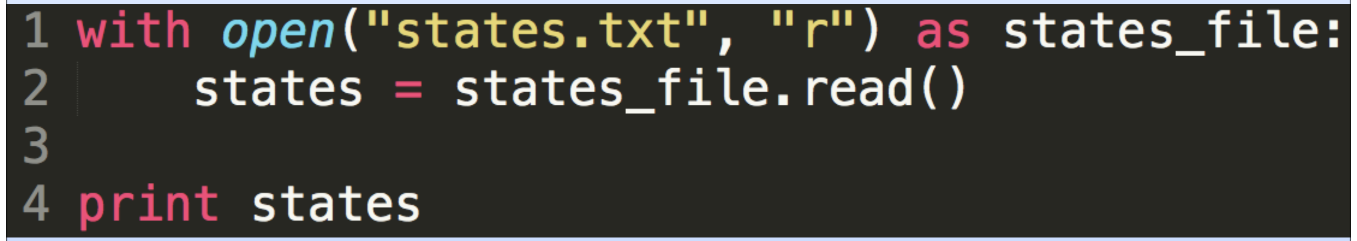
```
with open("states.txt", "r") as states_file:
    states = states_file.read()

print states
```

open( ) built-in function, tells Python to open a file.

Argument 1: The file you want to open, using relative paths

This slide is intentionally duplicate — reemphasize the points made here now that we've de-jargoned.

FILE HANDLING: MOST COMMON SYNTAX



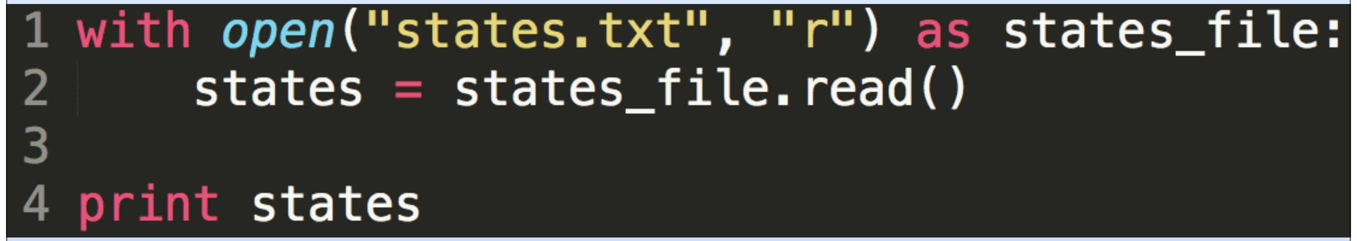
```python
with open("states.txt", "r") as states_file:
    states = states_file.read()

print states
```

open( ) built-in function, tells Python to open a file.

Argument 2: The "mode" to open the file in, as a string

- **r**: read-only mode
- **w**: write mode
- **a**: append mode

FILE HANDLING: MOST COMMON SYNTAX

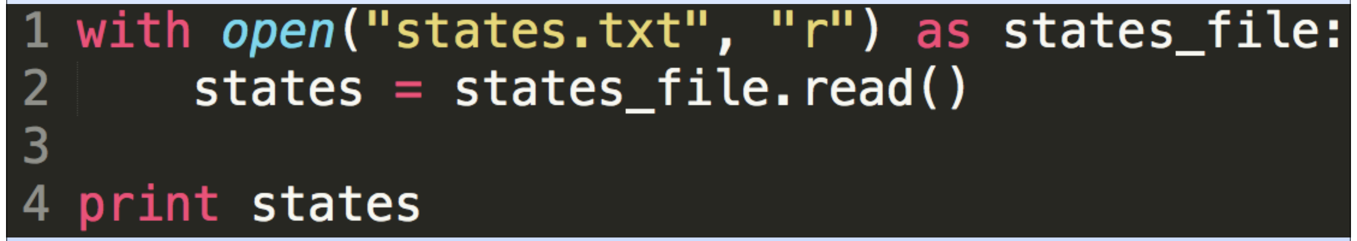


```
with open("states.txt", "r") as states_file:
    states = states_file.read()

print states
```

The as keyword creates a variable for your file handler.

The variable in this example is **states_file**, but you could use any variable name you want.

FILE HANDLING: MOST COMMON SYNTAX

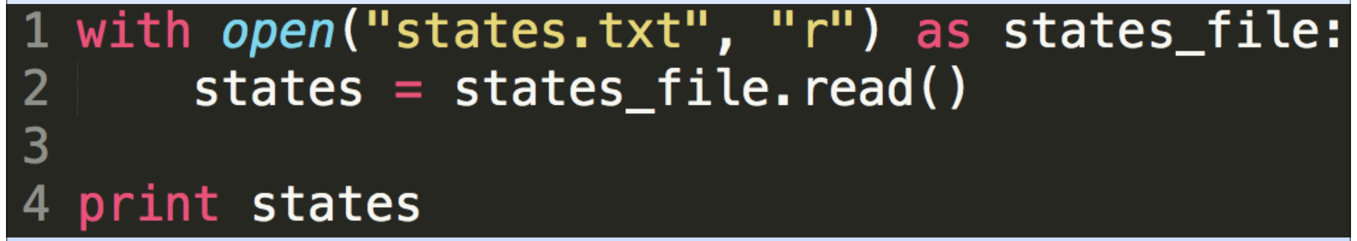


```
1 with open("states.txt", "r") as states_file:
2     states = states_file.read()
3
4 print states
```

.read( ) is a file method — a function that only works with file handlers. In this example, the file handler is **states_file.**

.read( ) will read the entire contents of the file. In line 2 above, I've saved it into the variable **states**.

FILE HANDLING: MOST COMMON SYNTAX



```
with open("states.txt", "r") as states_file:
    states = states_file.read()

print states
```

Outcome:

1. Open a file (**states.txt**)
2. Create a variable called **states** that has the entire contents of the file **states.txt**

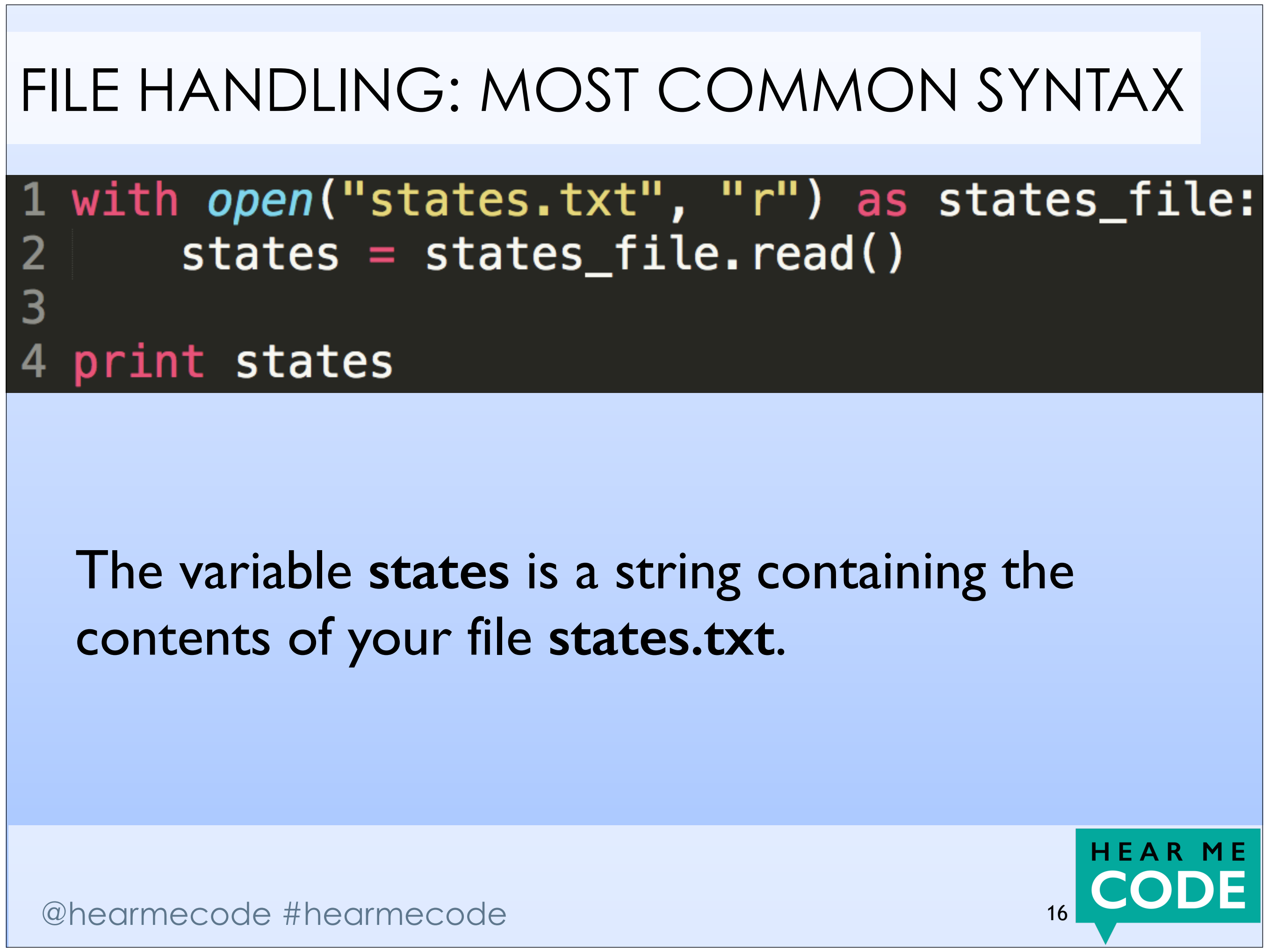
Hey Python, can you open up this file? Great! Can you save the full contents of this file into a variable? Awesome, thanks!

LET'S TRY IT OUT

- In the **python-lessons** repo, go to **section_07_(files)**
- Copy/paste or save **states.txt** onto your computer, in the same folder as your scripts.
- Write a script to open **states.txt** and print the contents of the file.

HEAR ME CODE

Show folks they can click directly on the states.txt link above to open it in a way that's easily copy-pasteable.



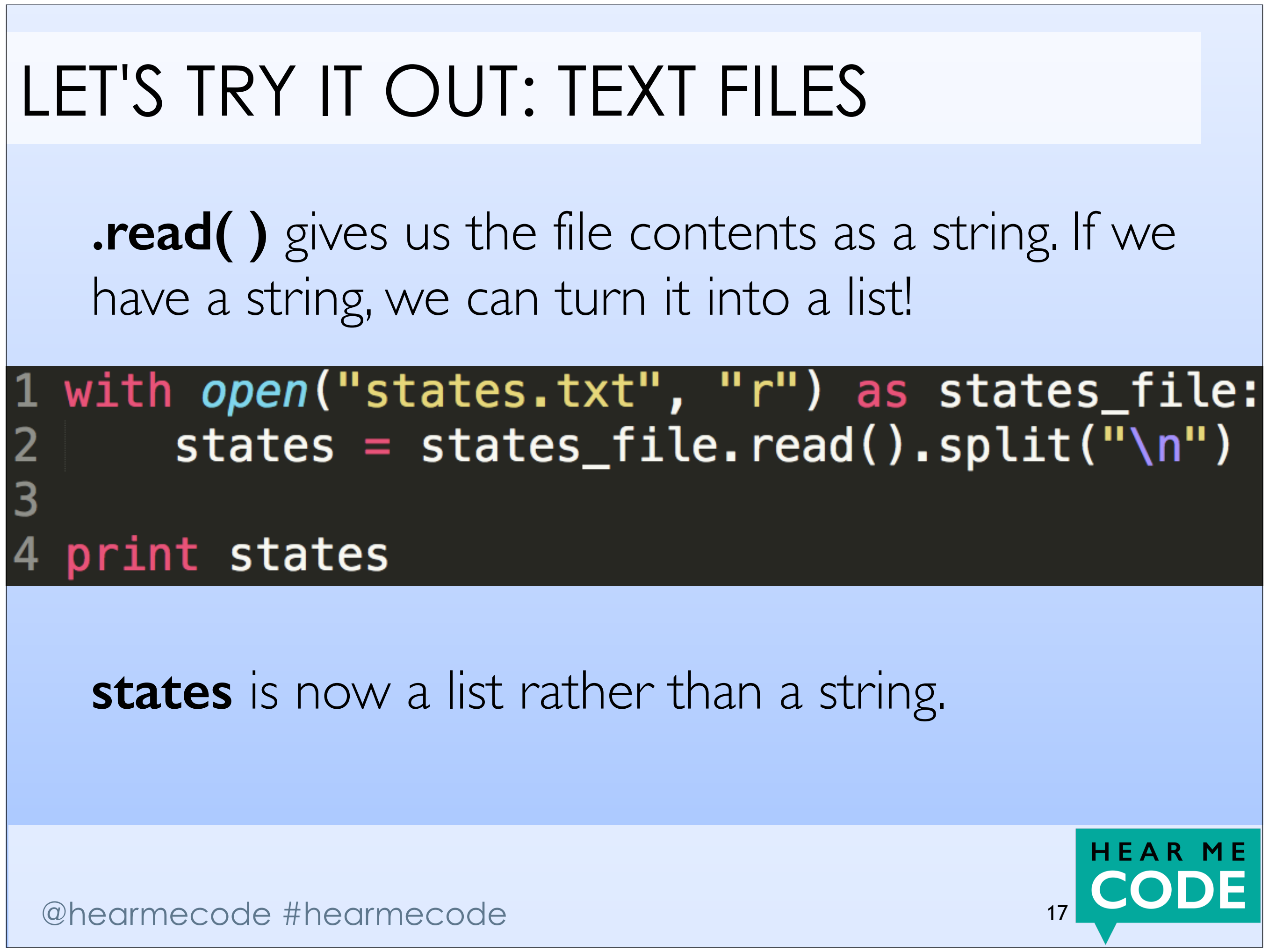
FILE HANDLING: MOST COMMON SYNTAX

```
with open("states.txt", "r") as states_file:
    states = states_file.read()

print states
```

The variable **states** is a string containing the contents of your file **states.txt**.

HEAR ME CODE

This is really just to recap everything we just did.



LET'S TRY IT OUT: TEXT FILES

.read( ) gives us the file contents as a string. If we have a string, we can turn it into a list!

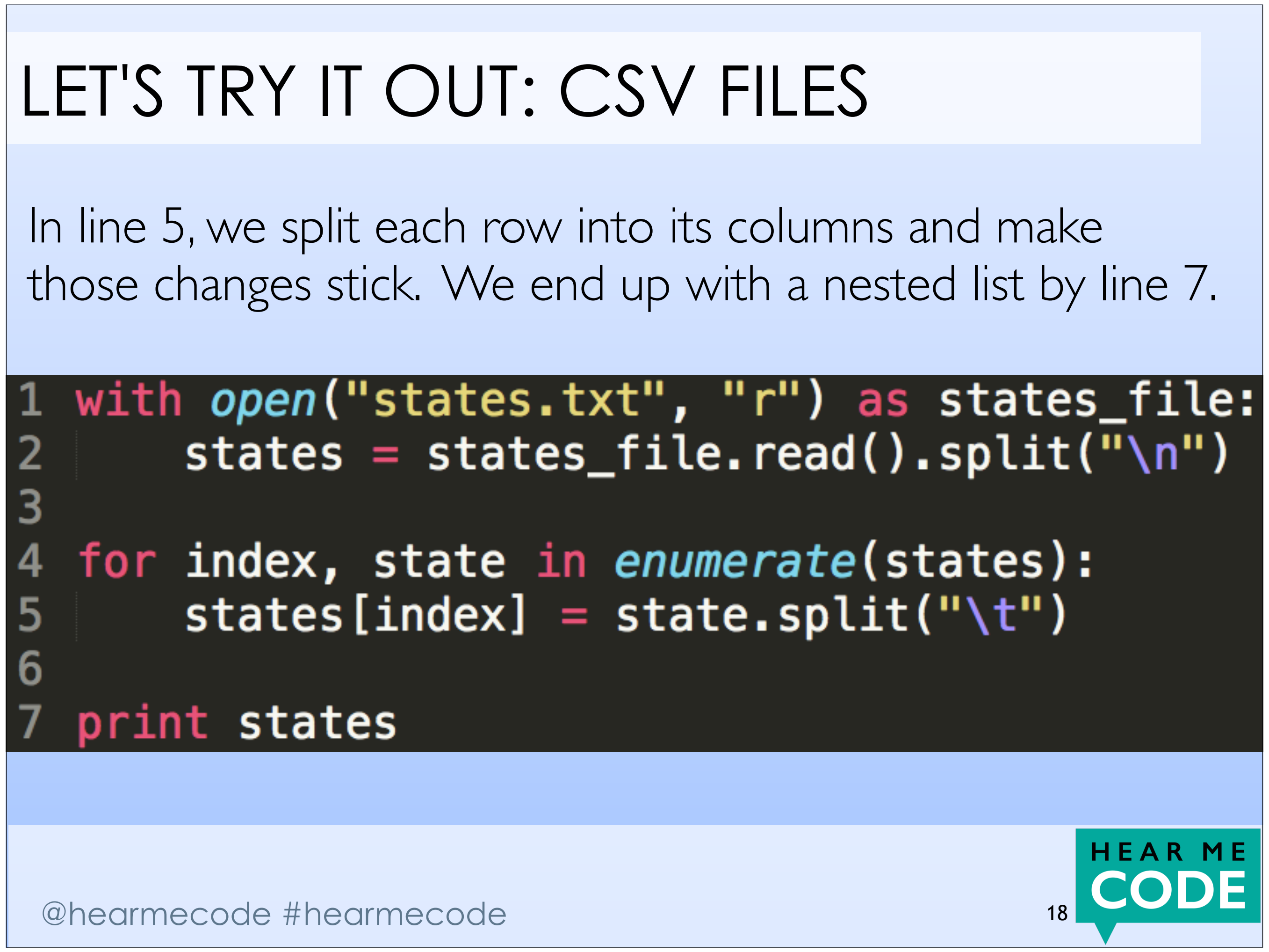
```
with open("states.txt", "r") as states_file:
    states = states_file.read().split("\n")

print states
```

states is now a list rather than a string.

Here, the only thing we changed from the previous example was adding

.split("\n")

Take the time to have everyone try it for themselves.



LET'S TRY IT OUT: CSV FILES

In line 5, we split each row into its columns and make those changes stick. We end up with a nested list by line 7.

```
1 with open("states.txt", "r") as states_file:
2     states = states_file.read().split("\n")
3
4 for index, state in enumerate(states):
5     states[index] = state.split("\t")
6
7 print states
```

Take a lot of time with lines 4 and 5. It's really challenging.

We need to use enumerate in order to get the position; then we overwrite the list as we loop through it.

At line 2, we split a string into a list; then item by item we loop through that list (the rows) and split out the columns and save it.

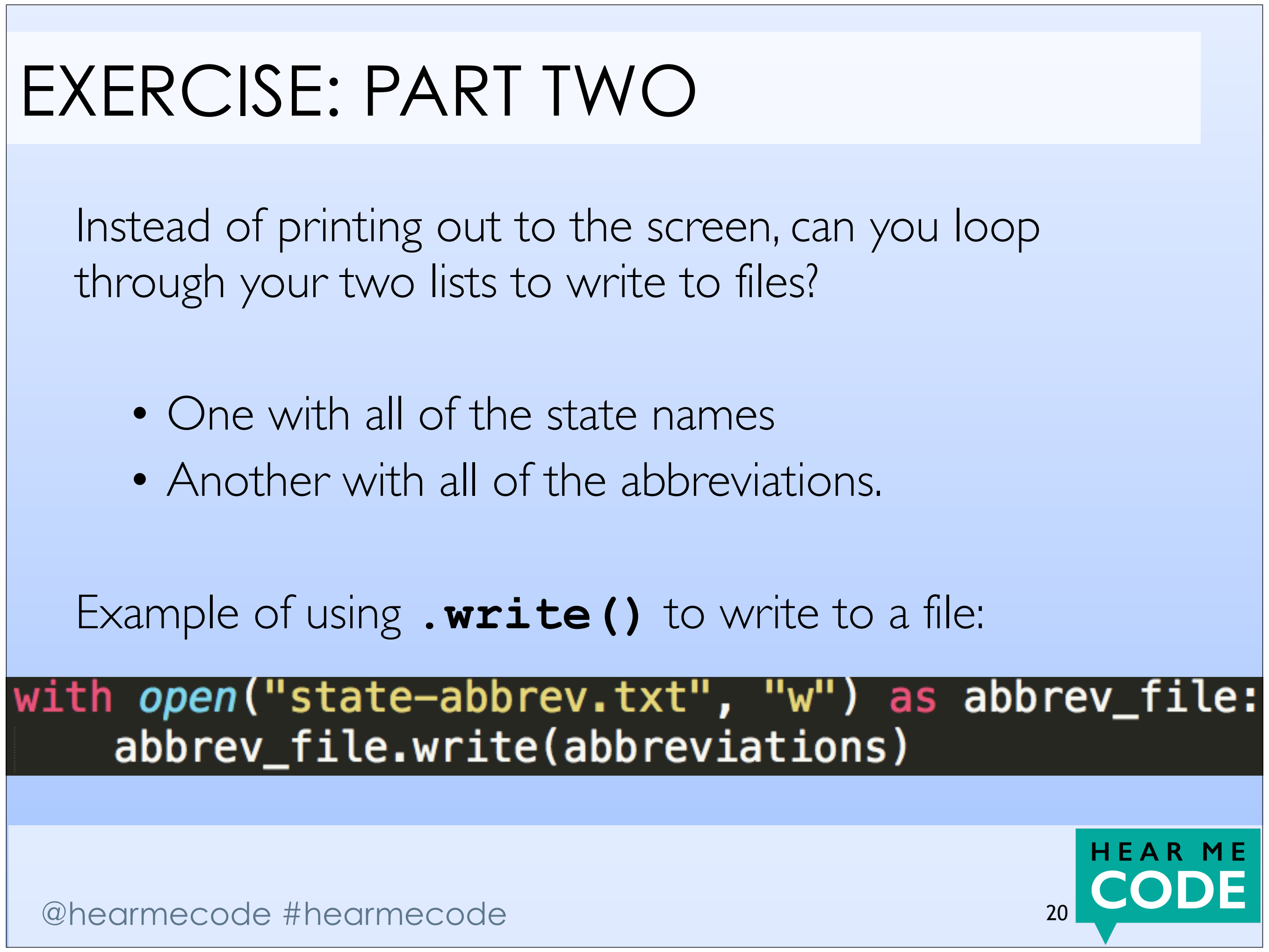
By line 7, we see that we have a nested list of lists.

EXERCISE: PART ONE

As in the previous slide, open either **states.txt** or **states.csv** and loop through to create two lists:

- One with all of the state names
- Another with all of the abbreviations.

Break everything into smaller steps, run and test often!



EXERCISE: PART TWO

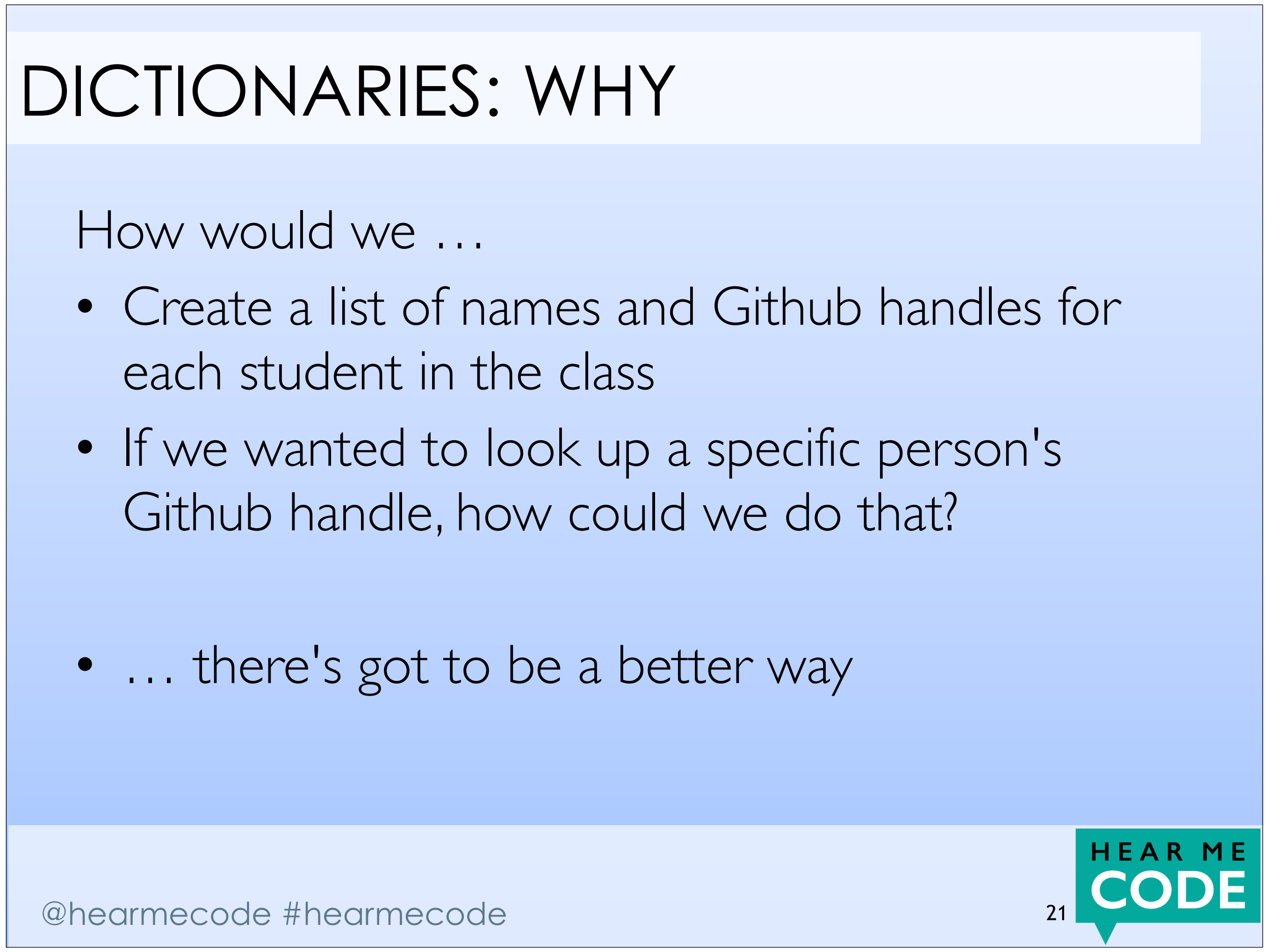
Instead of printing out to the screen, can you loop through your two lists to write to files?

- One with all of the state names
- Another with all of the abbreviations.

Example of using **`.write()`** to write to a file:

```
with open("state-abbrev.txt", "w") as abbrev_file:
    abbrev_file.write(abbreviations)
```

Note that you'll need the "w" flag (second parameter) to write to a file.



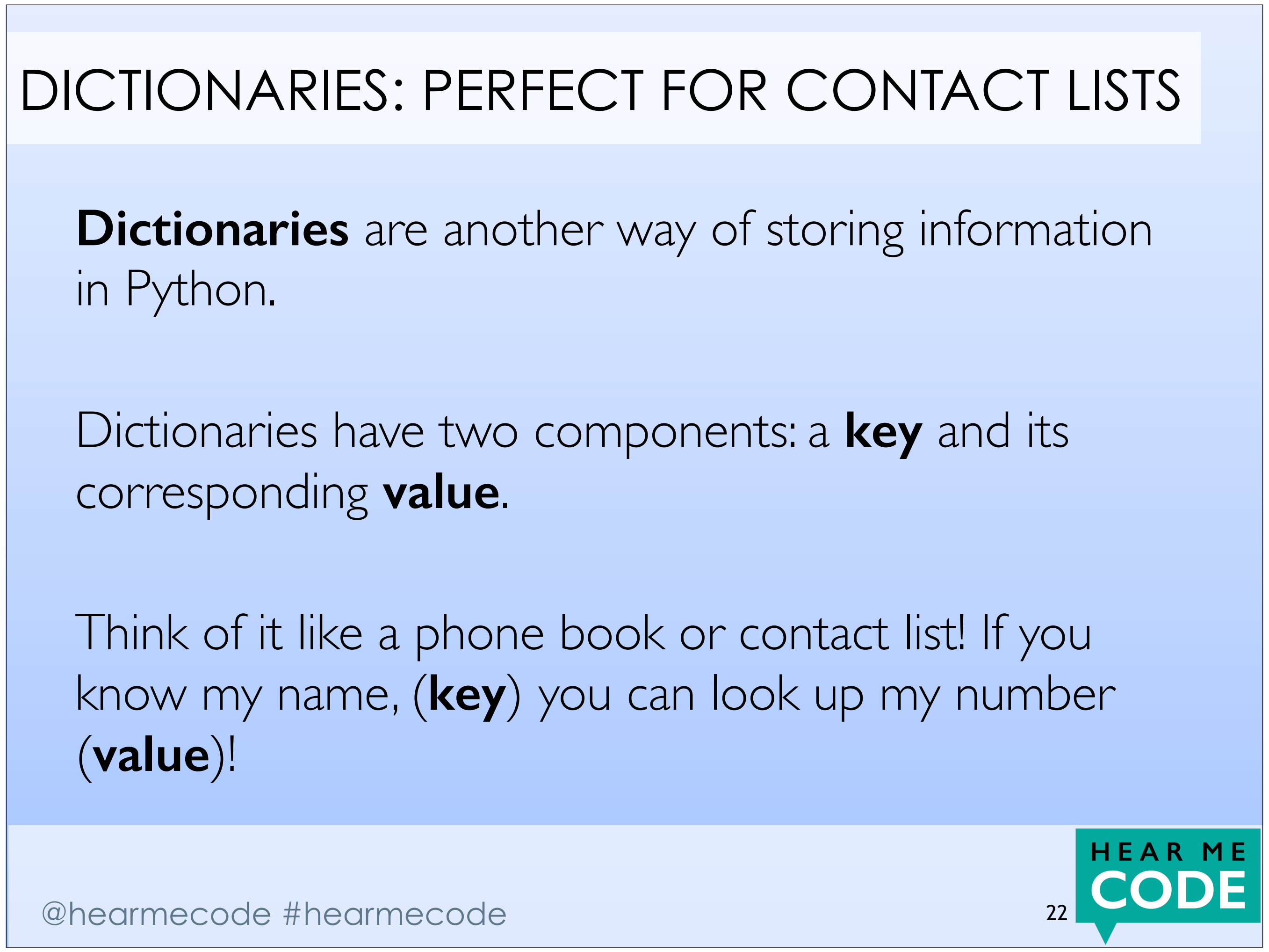
DICTIONARIES: WHY

How would we …

- Create a list of names and Github handles for each student in the class
- If we wanted to look up a specific person's Github handle, how could we do that?
- … there's got to be a better way

@hearmecode #hearmecode

Open up Sublime Text and start to create a list. Talk through how to do this without dictionaries, we'd have to loop through every single item and check to see if it's the one we're looking for.



DICTIONARIES: PERFECT FOR CONTACT LISTS

Dictionaries are another way of storing information in Python.

Dictionaries have two components: a **key** and its corresponding **value**.

Think of it like a phone book or contact list! If you know my name, (**key**) you can look up my number (**value**)!

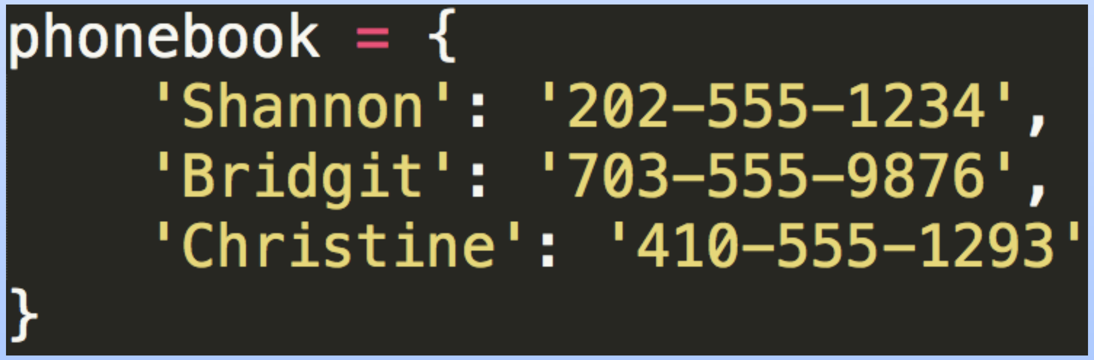
Throwaway joke: eventually no one will know what a "phonebook" is.

DICTIONARIES: SYNTAX

Creating an empty dictionary:

```
phonebook = {}
```

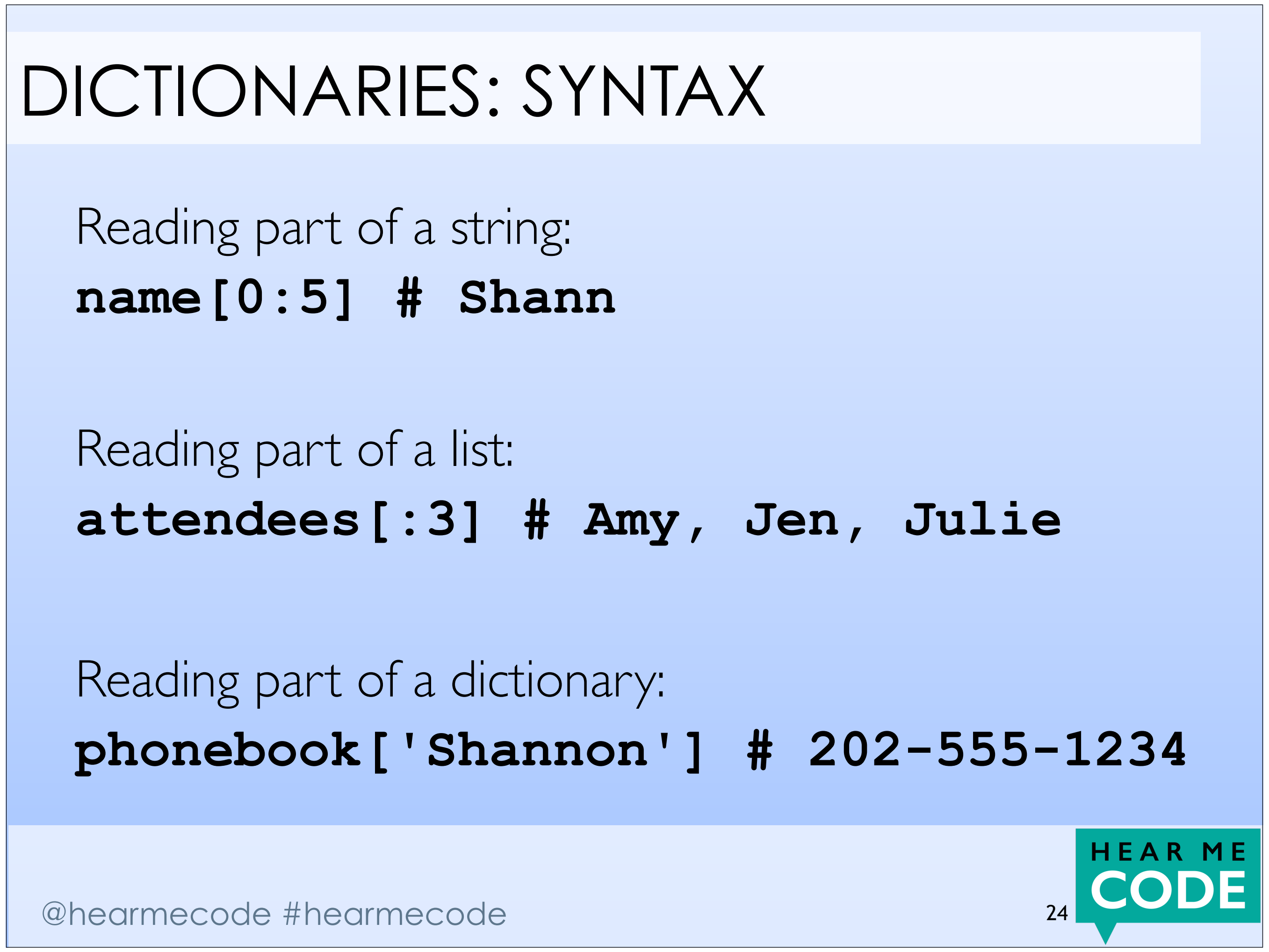
Creating a dictionary with items in it:



```
phonebook = {
    'Shannon': '202-555-1234',
    'Bridgit': '703-555-9876',
    'Christine': '410-555-1293'
}
```

HEAR ME CODE

Use your hands and reinforce key:value. If you know my name, you can look up my phone number.



DICTIONARIES: SYNTAX

Reading part of a string:

```
name[0:5] # Shann
```

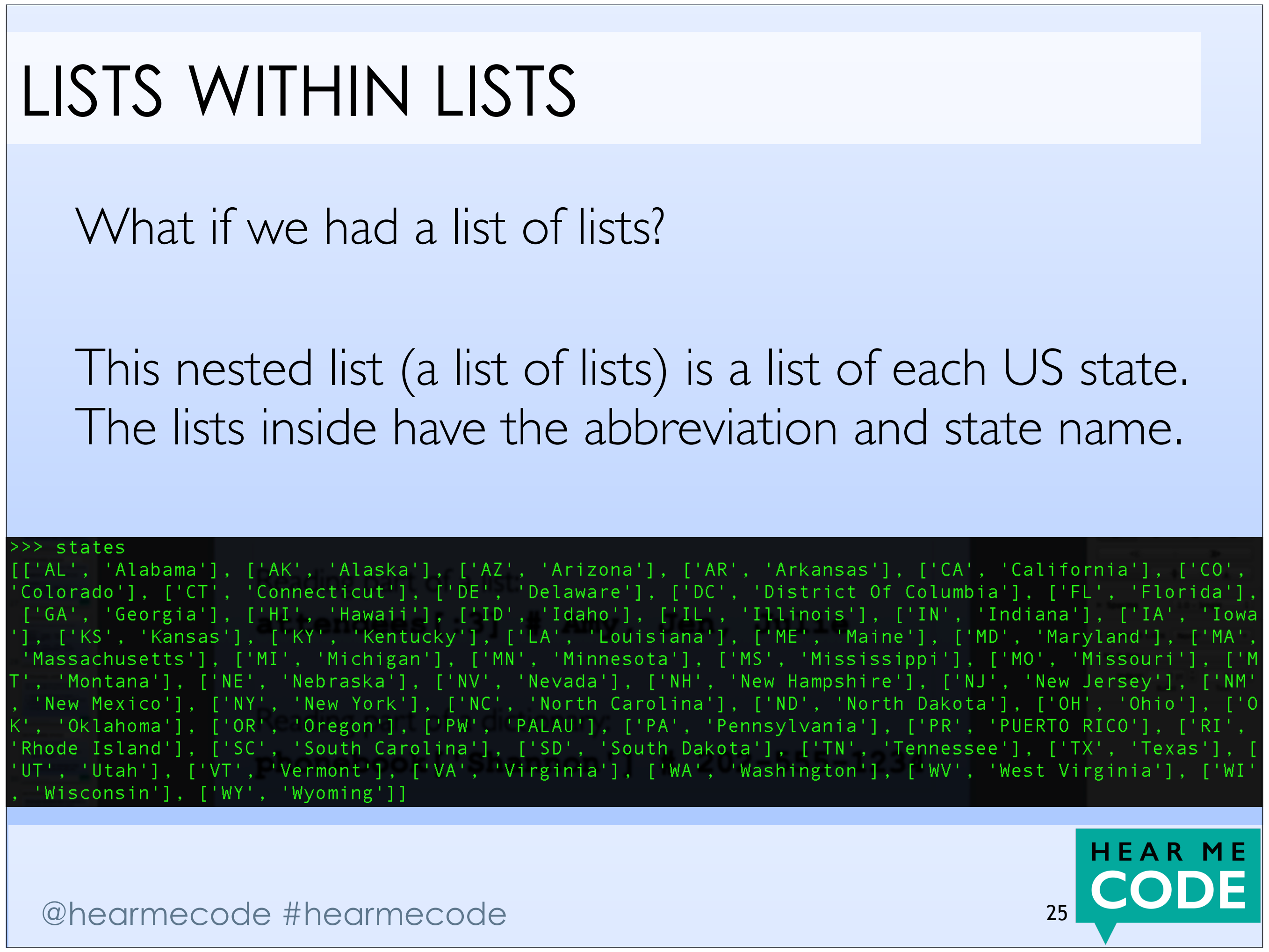
Reading part of a list:

```
attendees[:3] # Amy, Jen, Julie
```

Reading part of a dictionary:

```
phonebook['Shannon'] # 202-555-1234
```

Emphasize the similarities between slicing a string, slicing a list, and accessing a dictionary. In each case, the syntax uses square brackets and allows us to see part of the larger whole.



LISTS WITHIN LISTS

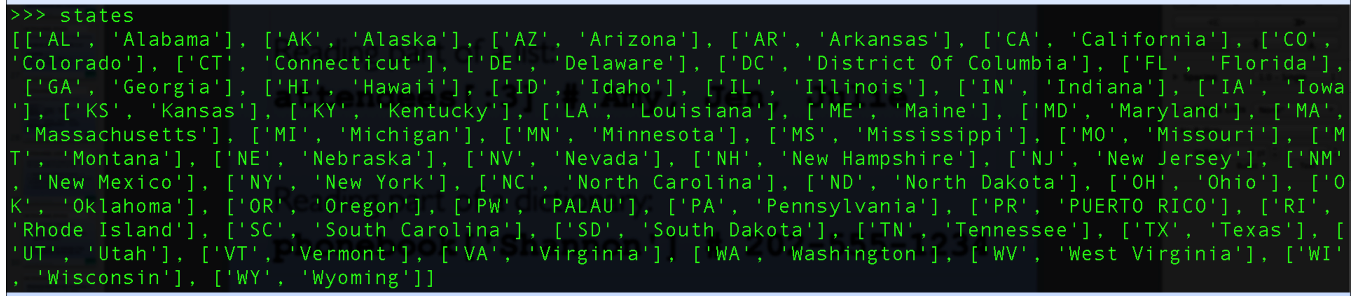
What if we had a list of lists?

This nested list (a list of lists) is a list of each US state. The lists inside have the abbreviation and state name.

```
>>> states
[['AL', 'Alabama'], ['AK', 'Alaska'], ['AZ', 'Arizona'], ['AR', 'Arkansas'], ['CA', 'California'], ['CO',
'Colorado'], ['CT', 'Connecticut'], ['DE', 'Delaware'], ['DC', 'District Of Columbia'], ['FL', 'Florida'],
 ['GA', 'Georgia'], ['HI', 'Hawaii'], ['ID', 'Idaho'], ['IL', 'Illinois'], ['IN', 'Indiana'], ['IA', 'Iowa
'], ['KS', 'Kansas'], ['KY', 'Kentucky'], ['LA', 'Louisiana'], ['ME', 'Maine'], ['MD', 'Maryland'], ['MA',
 'Massachusetts'], ['MI', 'Michigan'], ['MN', 'Minnesota'], ['MS', 'Mississippi'], ['MO', 'Missouri'], ['M
T', 'Montana'], ['NE', 'Nebraska'], ['NV', 'Nevada'], ['NH', 'New Hampshire'], ['NJ', 'New Jersey'], ['NM'
, 'New Mexico'], ['NY', 'New York'], ['NC', 'North Carolina'], ['ND', 'North Dakota'], ['OH', 'Ohio'], ['O
K', 'Oklahoma'], ['OR', 'Oregon'], ['PW', 'PALAU'], ['PA', 'Pennsylvania'], ['PR', 'PUERTO RICO'], ['RI',
'Rhode Island'], ['SC', 'South Carolina'], ['SD', 'South Dakota'], ['TN', 'Tennessee'], ['TX', 'Texas'], [
'UT', 'Utah'], ['VT', 'Vermont'], ['VA', 'Virginia'], ['WA', 'Washington'], ['WV', 'West Virginia'], ['WI'
, 'Wisconsin'], ['WY', 'Wyoming']]
```

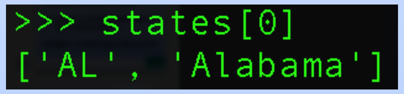
This seems like an abrupt pivot, but it's important to understanding nested dictionaries, too.

LISTS WITHIN LISTS



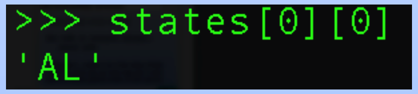
```
>>> states
[['AL', 'Alabama'], ['AK', 'Alaska'], ['AZ', 'Arizona'], ['AR', 'Arkansas'], ['CA', 'California'], ['CO',
'Colorado'], ['CT', 'Connecticut'], ['DE', 'Delaware'], ['DC', 'District Of Columbia'], ['FL', 'Florida'],
 ['GA', 'Georgia'], ['HI', 'Hawaii'], ['ID', 'Idaho'], ['IL', 'Illinois'], ['IN', 'Indiana'], ['IA', 'Iowa
'], ['KS', 'Kansas'], ['KY', 'Kentucky'], ['LA', 'Louisiana'], ['ME', 'Maine'], ['MD', 'Maryland'], ['MA',
 'Massachusetts'], ['MI', 'Michigan'], ['MN', 'Minnesota'], ['MS', 'Mississippi'], ['MO', 'Missouri'], ['M
T', 'Montana'], ['NE', 'Nebraska'], ['NV', 'Nevada'], ['NH', 'New Hampshire'], ['NJ', 'New Jersey'], ['NM'
, 'New Mexico'], ['NY', 'New York'], ['NC', 'North Carolina'], ['ND', 'North Dakota'], ['OH', 'Ohio'], ['O
K', 'Oklahoma'], ['OR', 'Oregon'], ['PW', 'PALAU'], ['PA', 'Pennsylvania'], ['PR', 'PUERTO RICO'], ['RI',
'Rhode Island'], ['SC', 'South Carolina'], ['SD', 'South Dakota'], ['TN', 'Tennessee'], ['TX', 'Texas'], [
'UT', 'Utah'], ['VT', 'Vermont'], ['VA', 'Virginia'], ['WA', 'Washington'], ['WV', 'West Virginia'], ['WI'
, 'Wisconsin'], ['WY', 'Wyoming']]
```

We're already familiar with how to view one item in this list.

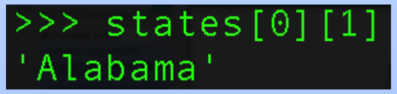


```
>>> states[0]
['AL', 'Alabama']
```

But states[0] is also a list! So to view one item in the states[0] list:



```
>>> states[0][0]
'AL'
```

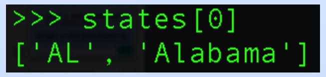


```
>>> states[0][1]
'Alabama'
```

LISTS WITHIN LISTS

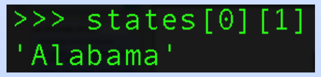
What type of object is **states**? A list.

What type is **states[0]**?



```
>>> states[0]
['AL', 'Alabama']
```

What type is **states[0][1]**?



```
>>> states[0][1]
'Alabama'
```

Can I slice those things to see a smaller part?

DICTIONARIES: SYNTAX

Reading part of a string:

```
name[0:5] # Shann
```

Reading part of a list:

```
attendees[:3] # Amy, Jen, Julie
```

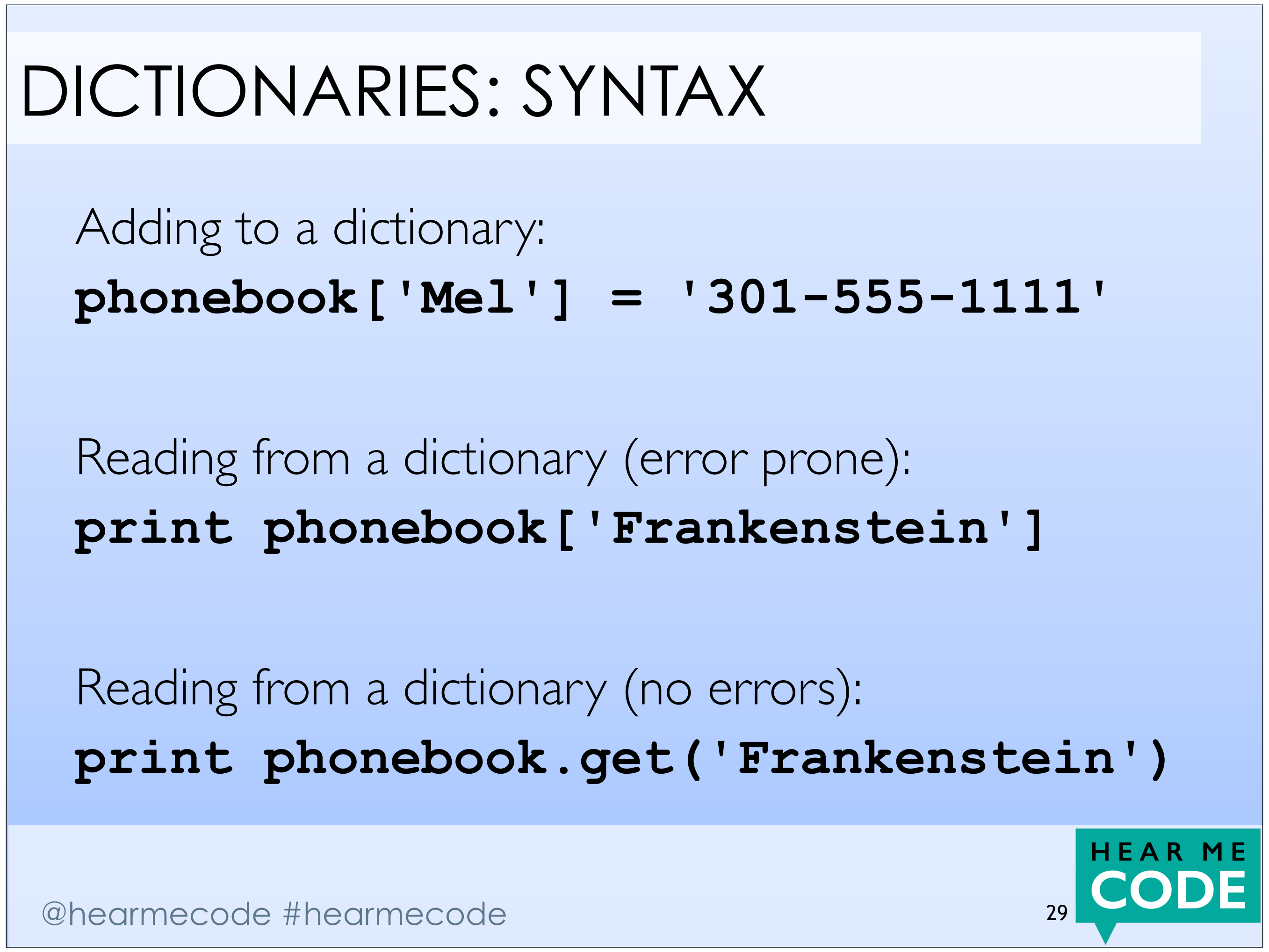
Reading part of a dictionary:

```
phonebook['Shannon'] # 202-555-1234
```

HEAR ME CODE

This slide duplicated for emphasis.

Emphasize the similarities between slicing a string, slicing a list, and accessing a dictionary. In each case, the syntax uses square brackets and allows us to see part of the larger whole.



DICTIONARIES: SYNTAX

Adding to a dictionary:

```
phonebook['Mel'] = '301-555-1111'
```

Reading from a dictionary (error prone):

```
print phonebook['Frankenstein']
```

Reading from a dictionary (no errors):

```
print phonebook.get('Frankenstein')
```

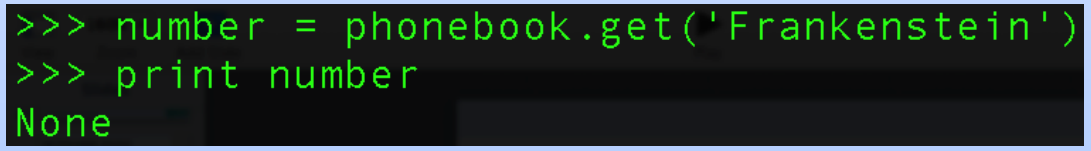
Have everyone try this out. What happens? Talk the error out.

Emphasize that getting a KeyError with a dictionary (I don't have this key) is similar to getting an IndexError with a list (I don't have this item)

WHAT'S NONE?

None is a special type in python, similar to **True** or **False.**

None is returned by the **.get( )** dictionary method when it couldn't find the key you're looking for.

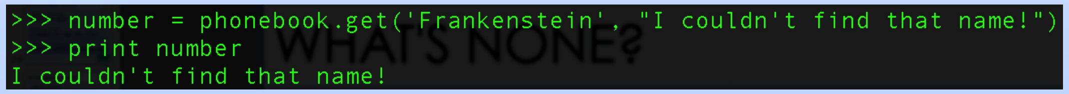


```
>>> number = phonebook.get('Frankenstein')
>>> print number
None
```

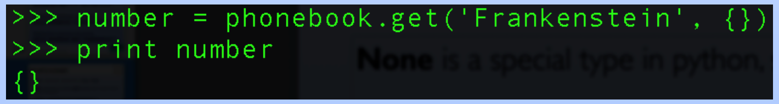
WHAT'S NONE?

By default, **.get( )** will give you **None** when it didn't find the key you were looking for.

But you can tell it to give you a different value — anything you want! A string, an empty dictionary, anything you can think of!



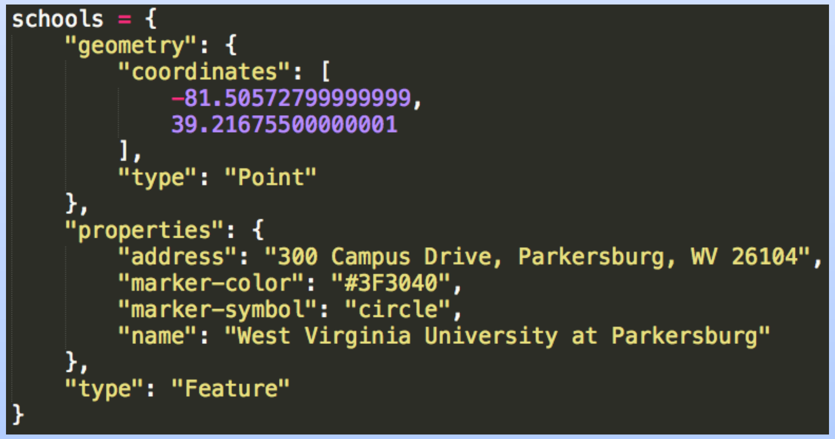
```
>>> number = phonebook.get('Frankenstein', "I couldn't find that name!")
>>> print number
I couldn't find that name!
```



```
>>> number = phonebook.get('Frankenstein', {})
>>> print number
{}
```

DICTIONARIES: SYNTAX

Dictionaries can contain strings, lists, or other dictionaries.

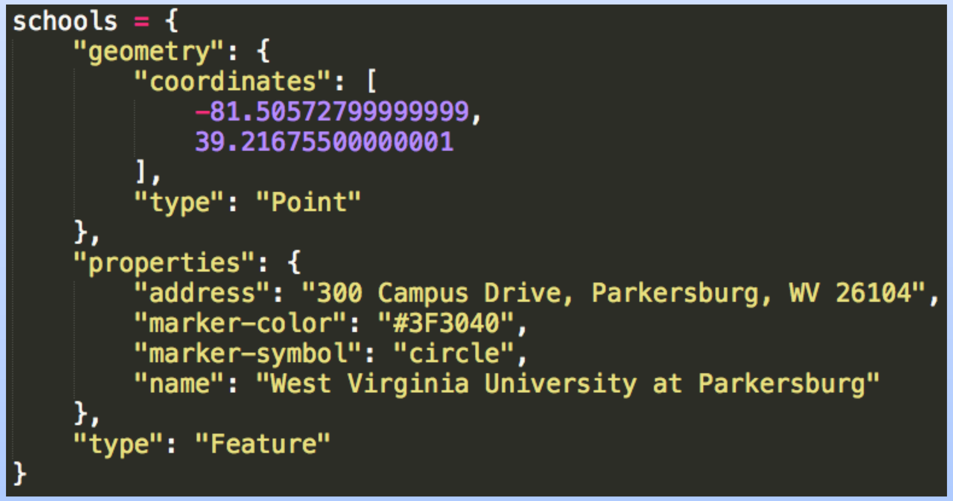


```
schools = {
    "geometry": {
        "coordinates": [
            -81.50572799999999,
            39.21675500000001
        ],
        "type": "Point"
    },
    "properties": {
        "address": "300 Campus Drive, Parkersburg, WV 26104",
        "marker-color": "#3F3040",
        "marker-symbol": "circle",
        "name": "West Virginia University at Parkersburg"
    },
    "type": "Feature"
}
```

Walk through each key / value pair and point out every single thing in this dictionary.

QUICK EXERCISE

Exercise instructions are here - open this link, save it to your computer, open it in Sublime/IDLE and work from there!

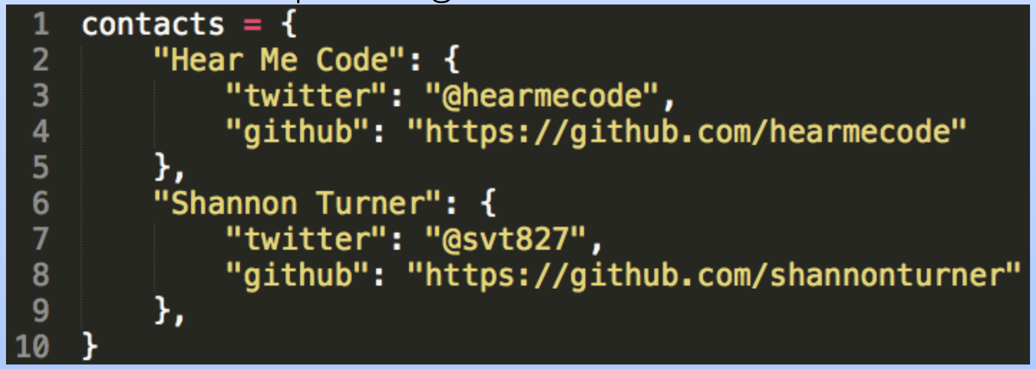


```
schools = {
    "geometry": {
        "coordinates": [
            -81.50572799999999,
            39.21675500000001
        ],
        "type": "Point"
    },
    "properties": {
        "address": "300 Campus Drive, Parkersburg, WV 26104",
        "marker-color": "#3F3040",
        "marker-symbol": "circle",
        "name": "West Virginia University at Parkersburg"
    },
    "type": "Feature"
}
```

EXERCISE

Exercise instructions are here - open this link, save it to your computer, open it in Sublime/IDLE and work from there!

Just do #1 for now. Once we've added items to our dictionary, we'll see how to loop through it in the next slides.



```
contacts = {
    "Hear Me Code": {
        "twitter": "@hearmecode",
        "github": "https://github.com/hearmecode"
    },
    "Shannon Turner": {
        "twitter": "@svt827",
        "github": "https://github.com/shannonturner"
    },
}
```

DICTIONARIES: LOOPING

Let's loop through the contacts list we just created. We have a handful of ways to do this.

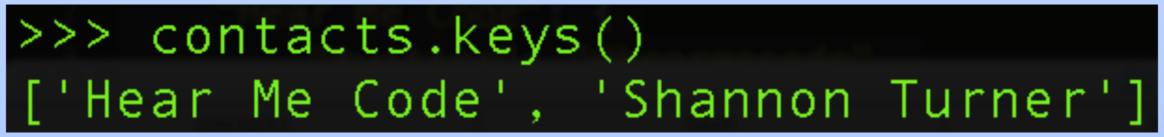
1. Looping by keys (Shannon, Hear Me Code, everyone else at your table…)

2. Looping by key / value pairs together

DICTIONARY METHODS

`.keys()` will create a list of all of the keys in your dictionary.

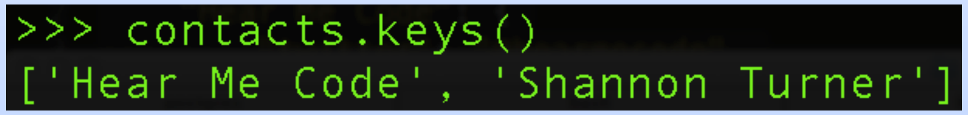
Because **dictionaries are unordered,** you might get keys in a different order than you see below, or a different order than you put them in. That's okay.



```
>>> contacts.keys()
['Hear Me Code', 'Shannon Turner']
```

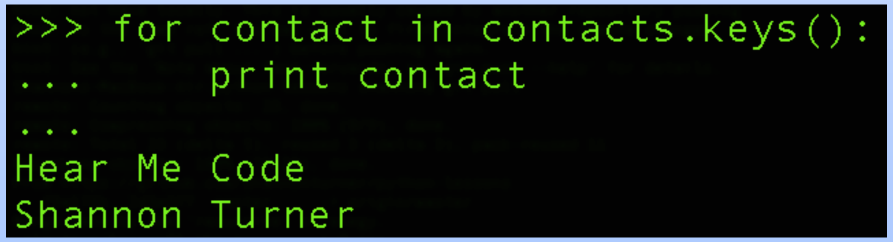
DICTIONARY METHODS

`.keys()` will create a list of all of the keys in your dictionary.



```
>>> contacts.keys()
['Hear Me Code', 'Shannon Turner']
```

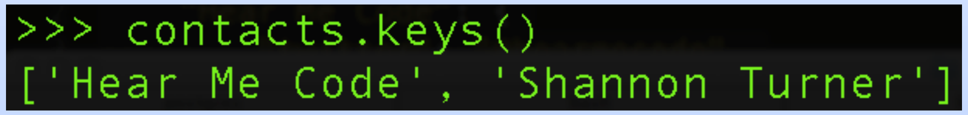
If you have a list, you can loop over it!



```
>>> for contact in contacts.keys():
...     print contact
...
Hear Me Code
Shannon Turner
```

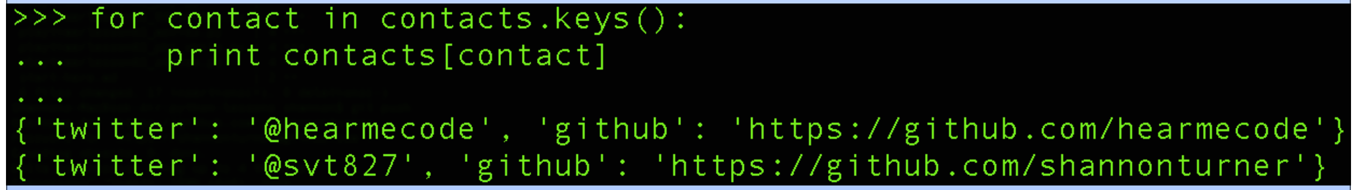
DICTIONARY METHODS

`.keys()` will create a list of all of the keys in your dictionary.

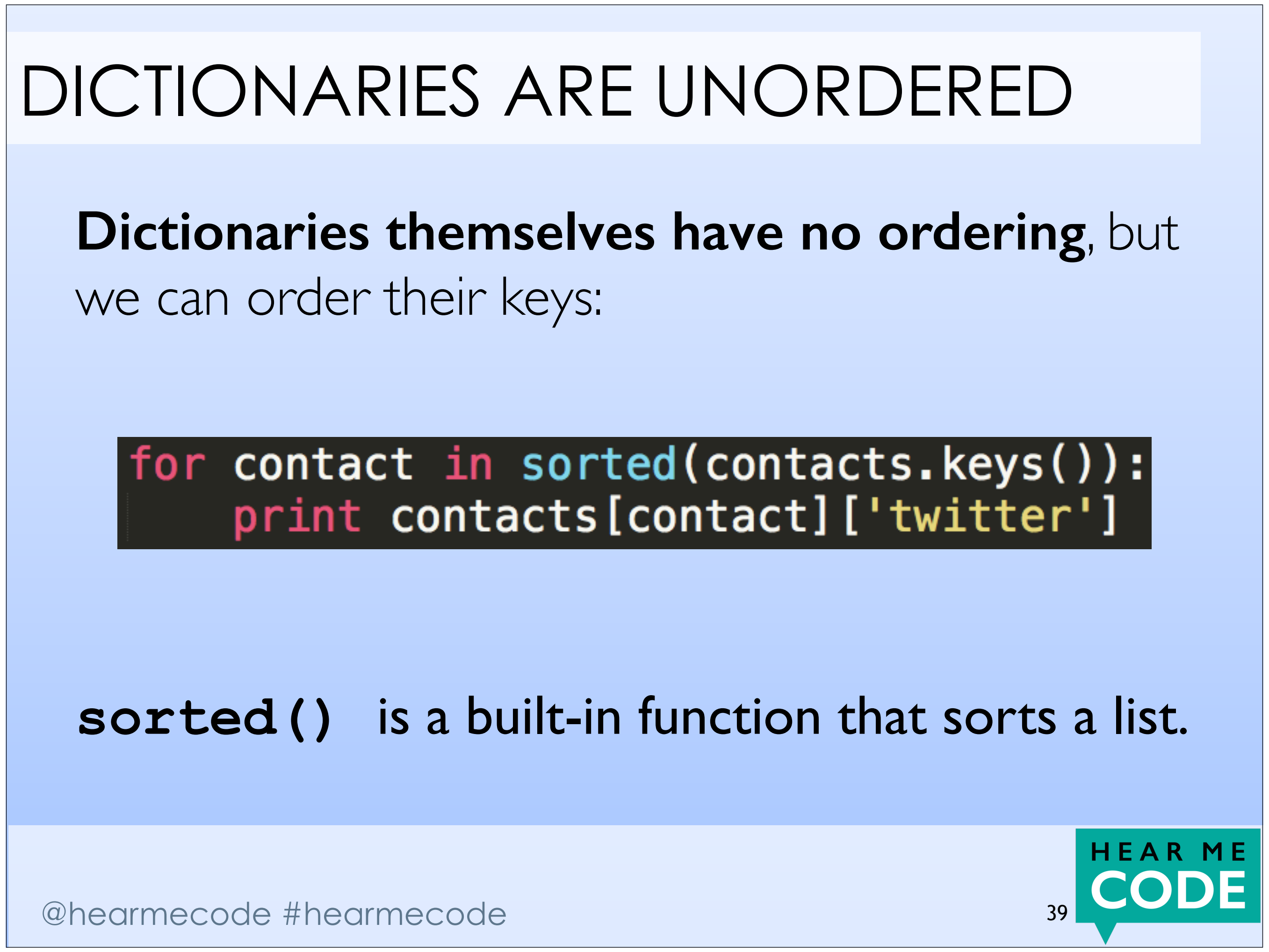


```
>>> contacts.keys()
['Hear Me Code', 'Shannon Turner']
```

If you have a list, you can loop over it!



```
>>> for contact in contacts.keys():
...     print contacts[contact]
...
{'twitter': '@hearmecode', 'github': 'https://github.com/hearmecode'}
{'twitter': '@svt827', 'github': 'https://github.com/shannonturner'}
```



DICTIONARIES ARE UNORDERED

Dictionaries themselves have no ordering, but we can order their keys:

```
for contact in sorted(contacts.keys()):
    print contacts[contact]['twitter']
```

`sorted()` is a built-in function that sorts a list.

Re-emphasize that you cannot rely on the ordering of a dictionary, you can only order lists of its keys.

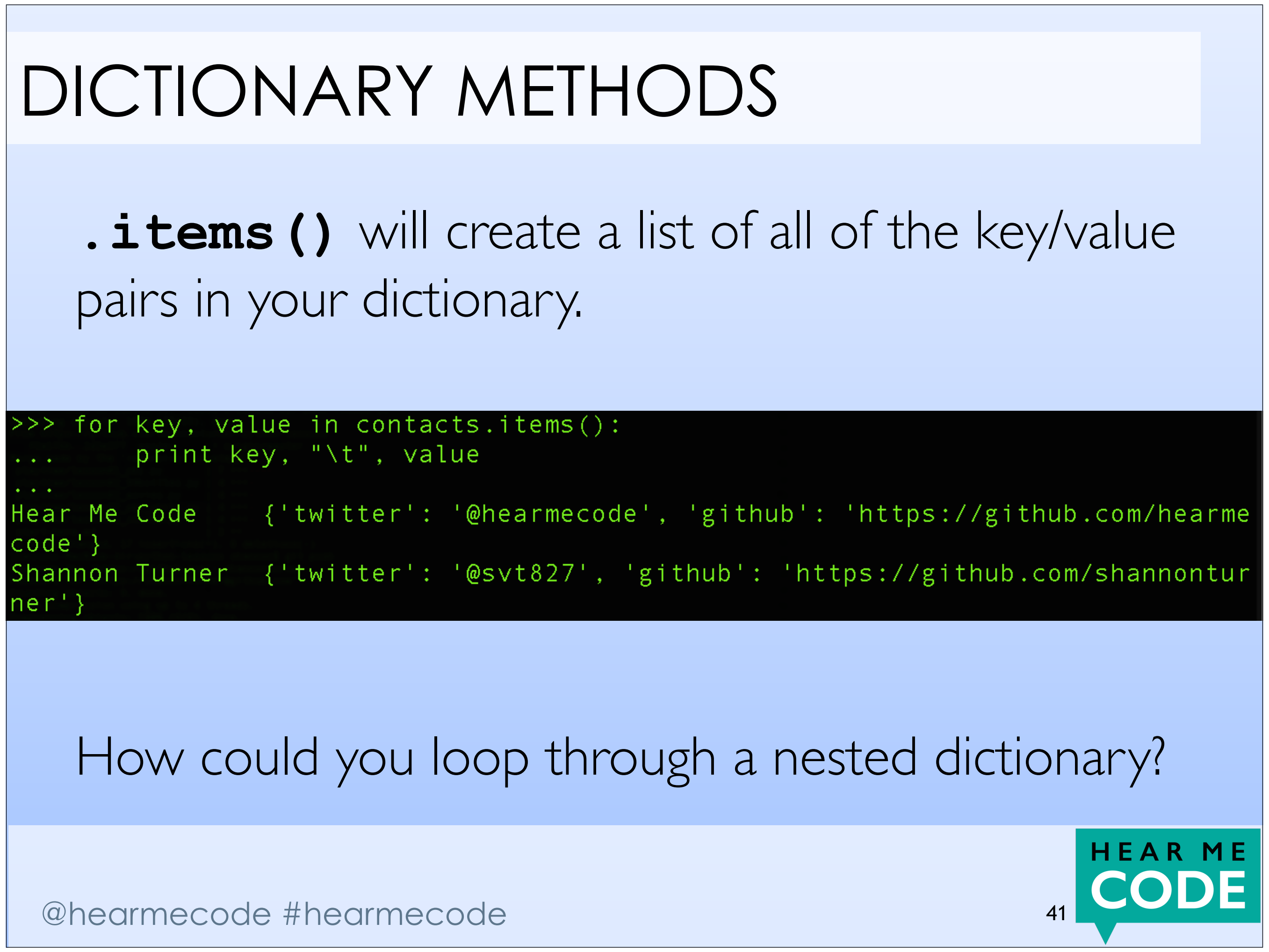
DICTIONARY METHODS

`.items()` will create a list of all of the key/value pairs in your dictionary.

```
>>> contacts.items()
[('Hear Me Code', {'twitter': '@hearmecode', 'github': 'https://github.com/hear
mecode'}), ('Shannon Turner', {'twitter': '@svt827', 'github': 'https://github.
com/shannonturner'})]
```

As with **`.keys()`**, if we have a list, we can loop over it. **`.items()`** gives us a list of lists!

Remind everyone that dictionaries are unordered, so you might see things in a different order and that's okay



DICTIONARY METHODS

`.items()` will create a list of all of the key/value pairs in your dictionary.

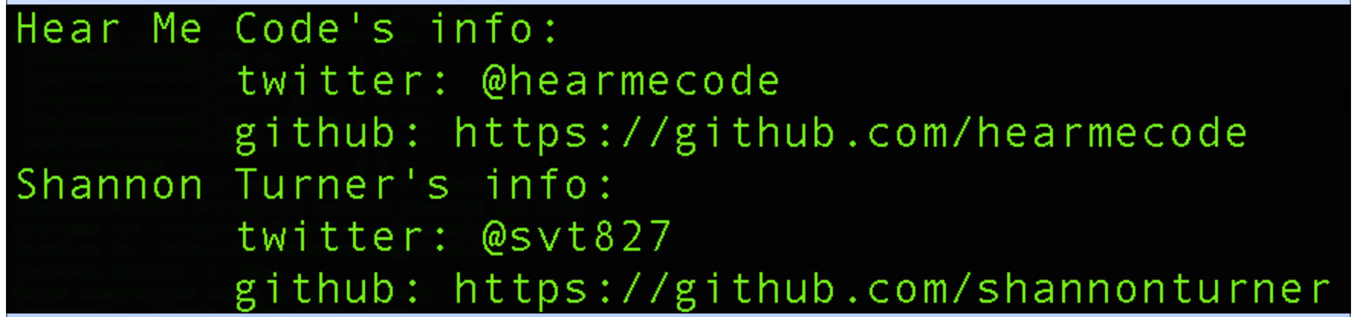
```
>>> for key, value in contacts.items():
...     print key, "\t", value
...
Hear Me Code    {'twitter': '@hearmecode', 'github': 'https://github.com/hearme
code'}
Shannon Turner  {'twitter': '@svt827', 'github': 'https://github.com/shannontur
ner'}
```

How could you loop through a nested dictionary?

Remind everyone that dictionaries are unordered, so you might see things in a different order and that's okay

EXERCISE: PART 2

Loop through the **contacts** dictionary to display everyone's contact information, like this:



```
Hear Me Code's info:
        twitter: @hearmecode
        github: https://github.com/hearmecode
Shannon Turner's info:
        twitter: @svt827
        github: https://github.com/shannonturner
```

Mention that not everyone will have a twitter; that's okay! Dictionaries can handle it :)

PLAYTIME!

Check out the **Hear Me Code slides** repo for practical examples, code samples, and more!

- Beginner: US States tables
- Beginner: Contacts list
- Advanced: Comparing two CSVs