FILE HANDLING AND DICTIONARIES

Shannon Turner

Twitter: @svt827

Github: http://github.com/shannonturner

OBJECTIVE

- Review Lesson Two
- Learn what file handling is
- 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

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"

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
```

FILE HANDLING: MOST COMMON SYNTAX

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

with keyword: Similar to a conditional by having an indentation level for code that follows.

File will be opened, commands are run, when all commands within indentation have been run, the file is closed automatically.

FILE HANDLING: MOST COMMON SYNTAX

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

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\alison\Desktop\clg\python

or

/Users/alison/Desktop/clg/python

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!

FILE HANDLING: MOST COMMON SYNTAX

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

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

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

FILE HANDLING: MOST COMMON SYNTAX

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

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
- **w**: write mode
- **a**: append mode

FILE HANDLING: MOST COMMON SYNTAX

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

The as keyword is used to create a variable out of 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

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

.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 the example above, I've saved it into the variable **states**.

FILE HANDLING: MOST COMMON SYNTAX

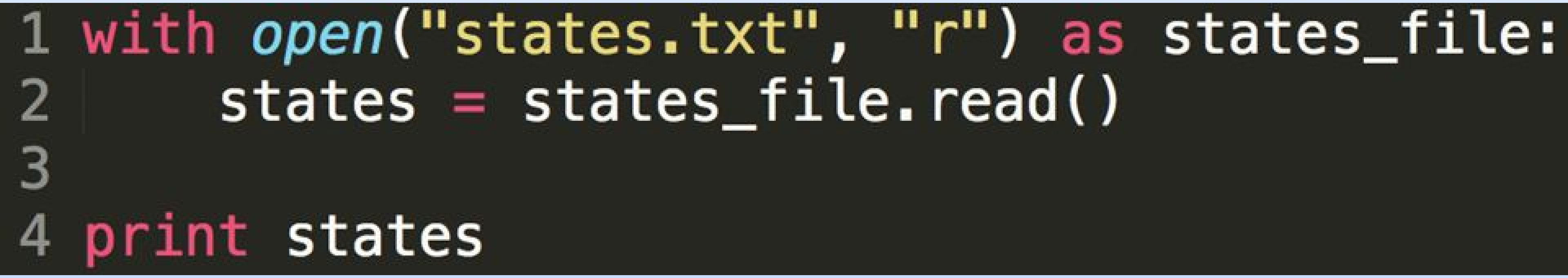
```
with open("states.txt", "r") as 
states_file:
    states = states_file.read()


print states
```

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

LET'S TRY IT OUT

- In the python-lessons-cny 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.

FILE HANDLING: MOST COMMON SYNTAX

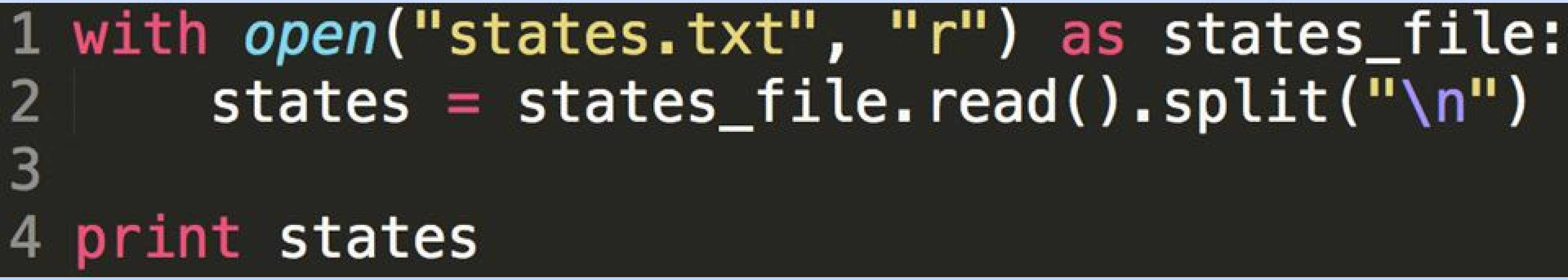


```
with open("states.txt", "r") as states_file:
    states = states_file.read()

print states
```

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

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!

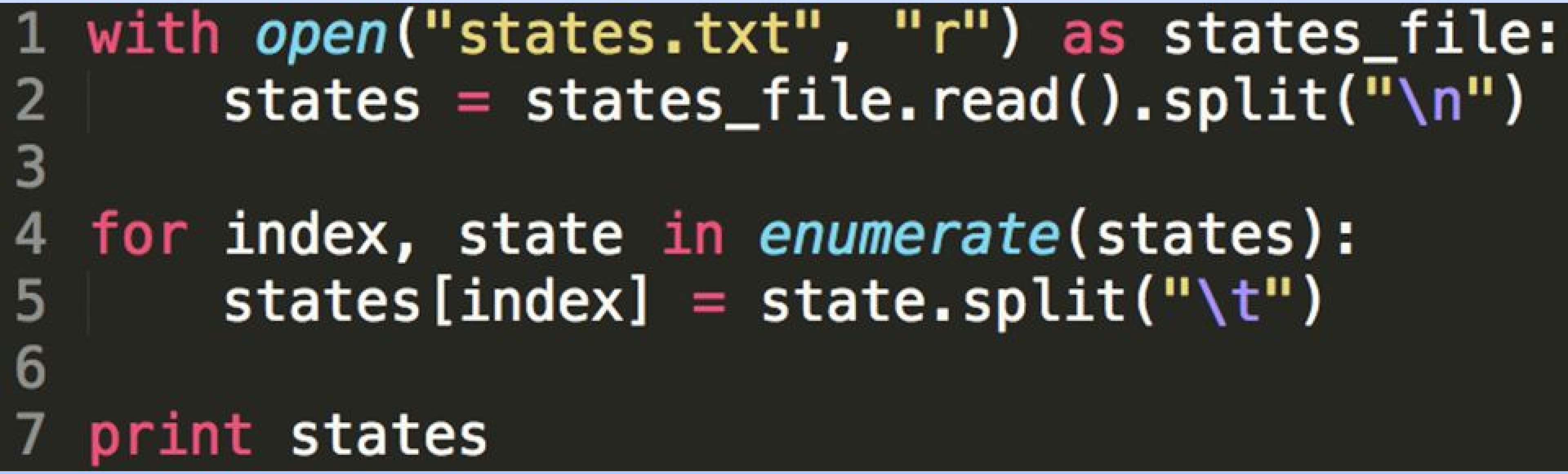


```
with open("states.txt", "r") as states_file:
    states = states_file.read().split("\n")

print states
```

states is now a list rather than a string.

LET'S TRY IT OUT: CSV FILES

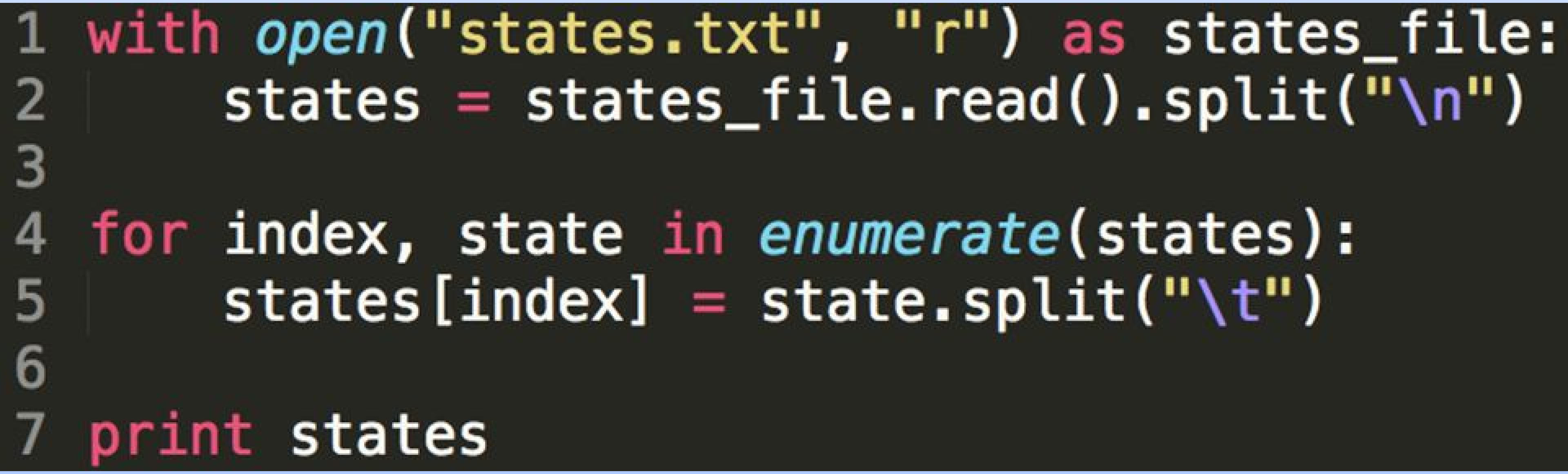
Here, we're splitting the file into a list at each line, and then individually splitting each line into a list by the commas.



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

for index, state in enumerate(states):
    states[index] = state.split("\t")

print states
```

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
```

EXERCISE

Goal 1: Change your Lesson 2 playtime states.py to use file handling to create your lists.

Open states.txt or states.csv

How to get the abbreviations into one list, and the state names into the other?

Break everything into smaller steps, run and test often!

EXERCISE

Goal 2: Instead of printing out to the screen, write to a file!

```
with open("states.html", "w") as new_file:

    new_file.write(some_string)
```

DICTIONARIES: WHY

How would we ...

- Create a list of names and Github names for each student in the class
- If we wanted to look up a specific person's Github name, how could we do that?

- ... there's got to be a better way

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)!

DICTIONARIES: SYNTAX

Creating an empty dictionary:

```
contacts = {}
```

Creating a dictionary with items in it:

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

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:

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

DICTIONARIES: SYNTAX

Adding to a dictionary:

```
contacts['Mel'] = '301-333-1212'
```

Reading from a dictionary (error prone):

```
print contacts['Frankenstein']
```

Reading from a dictionary (no errors):

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

DICTIONARIES: SYNTAX

Dictionaries can contain strings, lists, or other dictionaries.

```
phonebook['Mel'] = '301-333-1212'
```

```
attendees = {
'Feb 1': ['Jen', 'Amy', …],
'Feb 2': ['Rachel', 'Aliya', …]
}
```

DICTIONARIES: SYNTAX

Dictionaries can contain strings, lists, or other dictionaries.

```
contacts = {
'Shannon': {'phone': '202-555-1234',
'twitter': '@svt827', 'github':
'@shannonturner' },

'Anupama': {'phone': '410-333-9876',
'twitter': '@iamtheanupama', 'github':
'@apillalamarri'}
}
```

DICTIONARIES: SYNTAX

Looping through contacts by keys:

```
for contact in contacts.keys():
    print contact
```

`.keys()` creates a list of the keys of that dictionary, which you can then loop over.

DICTIONARIES: LOOPING

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

- Looping by keys (Shannon, Anupama, ...)
- Looping by values (all of Shannon's details, all of Anupama's details, ...)
- Looping by key / value pairs together

DICTIONARIES: SYNTAX

Dictionaries themselves have no ordering, but we can order their keys (or their values):

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

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

DICTIONARIES: SYNTAX

Looping through contacts by values:

```
for info in contacts.values():
    print info
```

`.values()` creates a list of the values of that dictionary, which you can then loop over. In this case, the phone number, github, twitter of each person.

DICTIONARIES: SYNTAX

Looping through contacts by key/value pairs:

```
for contact, info in contacts.items():
    print contact, info
```

`.items()` creates a nested list of the key and values of that dictionary, which you can then loop over.

EXERCISE: PART 1

Create a nested dictionary of contact info for everyone at your table.

```
contacts = {
'Shannon': {'phone': '202-555-1234',
'twitter': '@svt827', 'github':
'@shannonturner' },

'Anupama': {'phone': '410-333-9876',
'twitter': '@iamtheanupama', 'github':
'@apillalamarri'}
}
```

EXERCISE: PART 2

Loop through that dictionary to display everyone's contact information, like this:

```
Shannon's contact information is:
  Phone: 202-555-1234
  Twitter: @svt827
  Github: @shannonturner
```

EXERCISES

On my Github's python-lessons repo, go to the playtime folder!