

FILE HANDLING AND DICTIONARIES

Shannon Turner
Twitter: @svthmc

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
 - List of attendees
 - List of days in the week
 - List of months in the year
- 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

LIGHTNING REVIEW

- **.append()** adds one item to the end of a list

```
months = ['January', 'February']  
months.append('March')  
print months
```

```
['January', 'February', 'March']
```

- **.pop()** removes one item from the end of a list

LIGHTNING REVIEW

- Use `.split()` on a string to turn it into a list

```
months = "Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec"  
months = months.split(",")
```

```
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',  
'Oct', 'Nov', 'Dec']
```

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

```
for name in attendees:  
    print name
```

LIGHTNING REVIEW

- **enumerate()** works with for loops to give you the **position** of the list item and the **list item itself** at the same time

```
for index, name in enumerate(attendees):  
    print index, name
```

LIGHTNING REVIEW

```
1 days = [  
2     'Monday',  
3     'Tuesday',  
4     'Wednesday',  
5     'Thursday',  
6     'Friday',  
7     'Saturday',  
8     'Sunday',  
9 ]  
10  
11 for day in days:  
12     print day
```

```
Monday  
Tuesday  
Wednesday  
Thursday  
Friday  
Saturday  
Sunday
```


LIGHTNING REVIEW

```
for index, day in enumerate(days):  
    print index, day
```

```
0 Monday  
1 Tuesday  
2 Wednesday  
3 Thursday  
4 Friday  
5 Saturday  
6 Sunday
```

LIGHTNING REVIEW

```
for index, day in enumerate(days):  
    print "index is: {}".format(index)  
    print "day is: {}".format(day)  
    print "days[index] is: {}".format(days[index])
```

```
index is: 0  
day is: Monday  
days[index] is: Monday  
index is: 1  
day is: Tuesday  
days[index] is: Tuesday
```

EDITING LISTS

- You can change individual list items if you know their position/slicing number.

```
class_dates = [  
    'January 27',  
    'February 24',  
    'March ??'  
]  
class_dates[2] = 'March 24'
```

EDITING LISTS WITH ENUMERATE

- You can use this concept with **enumerate()** to edit every list item:

```
for index, date in enumerate(class_dates):  
    class_dates[index] = '{0}, 2018'.format(date)  
  
print class_dates
```

```
['January 27, 2018', 'February 24, 2018', 'March 24, 2018']
```

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

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

FILE HANDLING: MOST COMMON SYNTAX

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

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

Argument 1: The file you want to open.

FILE HANDLING: MOST COMMON SYNTAX

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

`open()` 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

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

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

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 Python script.
- Write a script to open **states.txt** and print the contents of the file.

FILE HANDLING: MOST COMMON SYNTAX

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

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

states is now a list rather than a string.

WE CAN OPEN SPREADSHEETS TOO

	A	B	C
1	Class	Date	Attendees
2	Lesson 1	February 24	44
3	Lesson 2	February 24	23
4	Lesson 3	February 24	12

CSV = COMMA SEPARATED VALUES

	A	B	C
1	Class	Date	Attendees
2	Lesson 1	February 24	44
3	Lesson 2	February 24	23
4	Lesson 3	February 24	12

```
1 Class,Date,Attendees
2 Lesson 1,February 24,44
3 Lesson 2,February 24,23
4 Lesson 3,February 24,12
```


WE START THE SAME AS WITH A TEXT FILE

- Let's open this CSV file and read from it

```
1 with open('class-stats.csv', 'r') as class_stats_file:  
2     class_stats = class_stats_file.read()  
3     print class_stats
```

```
Class,Date,Attendees  
Lesson 1,February 24,44  
Lesson 2,February 24,23  
Lesson 3,February 24,12
```

THEN SPLIT ON THE NEWLINES

- Only lines 4 and 5 are new.

```
1 with open('class-stats.csv', 'r') as class_stats_file:
2     class_stats = class_stats_file.read()
3     # print class_stats
4     class_stats = class_stats.split('\n')
5     print class_stats
```

```
['Class,Date,Attendees', 'Lesson 1,February 24,44', 'Lesson
2,February 24,23', 'Lesson 3,February 24,12']
```

ANOTHER WAY TO PICTURE IT

```
1 with open('class-stats.csv', 'r') as class_stats_file:
2     class_stats = class_stats_file.read()
3     # print class_stats
4     class_stats = class_stats.split('\n')
5     print class_stats
```

Class	Date	Attendees
Lesson 1	February 24	44
Lesson 2	February 24	23
Lesson 3	February 24	12

CSV = COMMA SEPARATED VALUES

- Only lines 7-10 are new.

```
1 with open('class-stats.csv', 'r') as class_stats_file:
2     class_stats = class_stats_file.read()
3     # print class_stats
4     class_stats = class_stats.split('\n')
5     # print class_stats
6
7     for index, stat in enumerate(class_stats):
8         class_stats[index] = stat.split(',')
9
10    print class_stats
```

```
[['Class', 'Date', 'Attendees'], ['Lesson 1', 'February 24', '44'],
['Lesson 2', 'February 24', '23'], ['Lesson 3', 'February 24', '12']]
```

LET'S TAKE A CLOSER LOOK

```
7     for index, stat in enumerate(class_stats):  
8         class_stats[index] = stat.split(',')
```

Class	Date	Attendees
-------	------	-----------

Lesson 1	February 24	44
----------	-------------	----

Lesson 2	February 24	23
----------	-------------	----

Lesson 3	February 24	12
----------	-------------	----

ANOTHER WAY TO PICTURE IT

```
1 with open('class-stats.csv', 'r') as class_stats_file:
2     class_stats = class_stats_file.read()
3     # print class_stats
4     class_stats = class_stats.split('\n')
5     # print class_stats
6
7     for index, stat in enumerate(class_stats):
8         class_stats[index] = stat.split(',')
9
10    print class_stats
```

Class	Date	Attendees
Lesson 1	February 24	44
Lesson 2	February 24	23
Lesson 3	February 24	12

HOW TO GET TO ITEMS IN A NESTED LIST

- `class_stats[0]` is a list
- `class_stats[0][0]` is a string

	[0]	[1]	[2]
<code>class_stats[0]</code>	Class	Date	Attendees
<code>class_stats[1]</code>	Lesson 1	February 24	44
<code>class_stats[2]</code>	Lesson 2	February 24	23
<code>class_stats[3]</code>	Lesson 3	February 24	12

WHY USE PYTHON FOR SPREADSHEETS?

- Compare attendance
 - Lesson and class size separated out
 - What is the average size of lesson 1?
 - What was the lowest class size?
 - The highest?
 - How many lesson 2s have I run?
 - What is the difference in class sizes between lessons?

WHY USE PYTHON FOR SPREADSHEETS?

- Given a spreadsheet of petition signatures, create a nicely-formatted document to send to the printer
- Bulk import records into a database
- Turn a spreadsheet of museum locations into a map:
<https://shannonvturner.com/museums>

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.csv', 'r') as states_file:
2     states = states_file.read().split('\n')
3
4 for index, state in enumerate(states):
5     states[index] = state.split(',')
6
7 print states
```

LISTS WITHIN LISTS

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

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

LISTS WITHIN LISTS

```
[['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'],
```

We're already familiar with how to view one item in a list:

```
8 print states[0]  
0  
['AL', 'Alabama']
```

LISTS WITHIN LISTS

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

```
8 print states[0][0]  
a  
AL
```

or

```
8 print states[0][1]  
a  
Alabama
```

LISTS WITHIN LISTS

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

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

```
8 print states[0]  
0  
['AL', 'Alabama']
```

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

```
8 print states[0][1]  
0  
Alabama
```

Can I slice those things to see a smaller part?

EXERCISE: PART ONE

Building from the previous slide, open **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

Loop through your two lists to write their contents to two files:

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

```
with open('state-abbrevs.txt', 'w') as abbrev_file:  
    for abbreviation in abbreviations:  
        abbrev_file.write(abbreviation)  
        abbrev_file.write('\n')
```


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

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**)!

CREATING A DICTIONARY

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'  
}
```

READING PART OF A DICTIONARY

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
```

ADDING TO A DICTIONARY

Add an item to a dictionary:

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

Dictionaries are **unordered**.

The order of your dictionary may change as you add or remove items!

REAL PHONEBOOKS ARE A BIT MORE COMPLEX

In your phone's contacts app, what fields might you find?

- Name
- Organization
- Phone number (and type)
- Email
- Address
- ... a whole lot more

REAL PHONEBOOKS ARE COMPLEX

```
1 contacts = {  
2     'Shannon Turner': {  
3         'organization': 'Hear Me Code',  
4         'phone': {  
5             'mobile': '202-555-1234'  
6         },  
7         'email': 'shannon@hearmecode.com',  
8         'address': {  
9             'address1': '123 SEA LANE SW',  
10            'address2': 'Apartment 987',  
11            'city': 'Washington',  
12            'state': 'DC',  
13            'zip': '20000'  
14        },  
15     },  
16 }
```

"SLICE" IT LIKE A NESTED LIST

We have a dictionary within a dictionary:

```
18 print contacts['Shannon Turner']  
19  
{'organization': 'Hear Me Code', 'address': {'address1': '123 SEA LANE SW',  
'address2': 'Apartment 987', 'state': 'DC', 'zip': '20000', 'city':  
'Washington'}, 'email': 'shannon@hearmecode.com', 'phone': {'mobile':  
'202-555-1234'}}
```

Just keep slicing:

```
19 print contacts['Shannon Turner']['organization']  
20  
Hear Me Code
```


DICTIONARIES CAN CONTAIN ...

... strings, lists, even 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"  
}
```

EXERCISE

Save these [exercise instructions](#) 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"  
}
```

DICTIONARIES ARE FLEXIBLE

We just met Carla, so we don't have as many details about her in our phonebook as we do for Shannon.

```
contacts['Carla Conference'] = {  
    'phone': {  
        'mobile': '301-555-0000'  
    },  
    'notes': 'I met her at a conference!'  
}
```

I DON'T HAVE THAT KEY

```
contacts['Carla Conference'] = {  
    'phone': {  
        'mobile': '301-555-0000'  
    },  
    'notes': 'I met her at a conference!'  
}
```

```
28 print contacts['Carla Conference']['organization']
```

Traceback (most recent call last):

File "/Users/shannon/Documents/L3-review/lesson3.py", line 28, in <module>

```
    print contacts['Carla Conference']['organization']
```

KeyError: 'organization'

USE .GET() TO KEEP GOING

When you get a KeyError, your program stops. That's not always what you want.

```
28 # If there is no 'organization' key, Python will stop with a KeyError
29 print contacts['Carla Conference']['organization']
30
31 # I'll get None if there is no 'organization' key:
32 print contacts['Carla Conference'].get('organization')
33
34 # Or I could specify a default value if the 'organization' key does not
   exist:
35 print contacts['Carla Conference'].get('organization', "I don't know
   their organization!")
```

EXERCISE

Save these [exercise instructions](#) 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.

```
1 contacts = {  
2     "Hear Me Code": {  
3         "twitter": "@hearmecode",  
4         "github": "https://github.com/hearmecode"  
5     },  
6     "Shannon Turner": {  
7         "twitter": "@svthmc",  
8         "github": "https://github.com/shannonturner"  
9     },  
10 }
```

LOOPING THROUGH A DICTIONARY

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

.KEYS() CREATES A LIST

.**keys**() creates a list of all of the keys in your dictionary.

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

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.

WE CAN LOOP OVER A LIST!

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

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

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

```
14 for name in contacts.keys():  
15     print name  
16  
Hear Me Code  
Shannon Turner
```

WE CAN LOOP OVER A LIST!

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

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

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

```
14 for name in contacts.keys():  
15     print contacts[name]  
16  
{'twitter': '@hearmecode', 'github': 'https://github.com/hearmecode'}  
{'twitter': '@svthmc', 'github': 'https://github.com/shannonturner'}
```

DICTIONARIES ARE UNORDERED

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

```
for name in sorted(contacts.keys()):  
    print contacts[name]
```

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

.ITEMS() CREATES A LIST OF KEY/VALUE PAIRS

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

```
12 print contacts.items()  
13  
[('Hear Me Code', {'twitter': '@hearmecode', 'github': 'https://github.com/hearmecode'}), ('Shannon Turner', {'twitter': '@svthmc', '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!

.ITEMS() CREATES A NESTED LIST

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

```
12 for key, value in contacts.items():  
13     print key, value
```

```
Hear Me Code {'twitter': '@hearmecode', 'github': 'https://github.com/  
hearmecode'}  
Shannon Turner {'twitter': '@svthmc', 'github': 'https://github.com/  
shannonturner'}
```

.ITEMS() CREATES A NESTED LIST

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

```
12 for name, details in contacts.items():  
13     print name, details
```

```
Hear Me Code {'twitter': '@hearmecode', 'github': 'https://github.com/  
hearmecode'}  
Shannon Turner {'twitter': '@svthmc', 'github': 'https://github.com/  
shannonturner'}
```

EXERCISE: PART 2

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

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

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](#)

WHAT NOW?

- [Lessons 4 & 5](#)
- Organize a study group on the listserv
- Come be a teaching assistant
- Practice, practice, practice