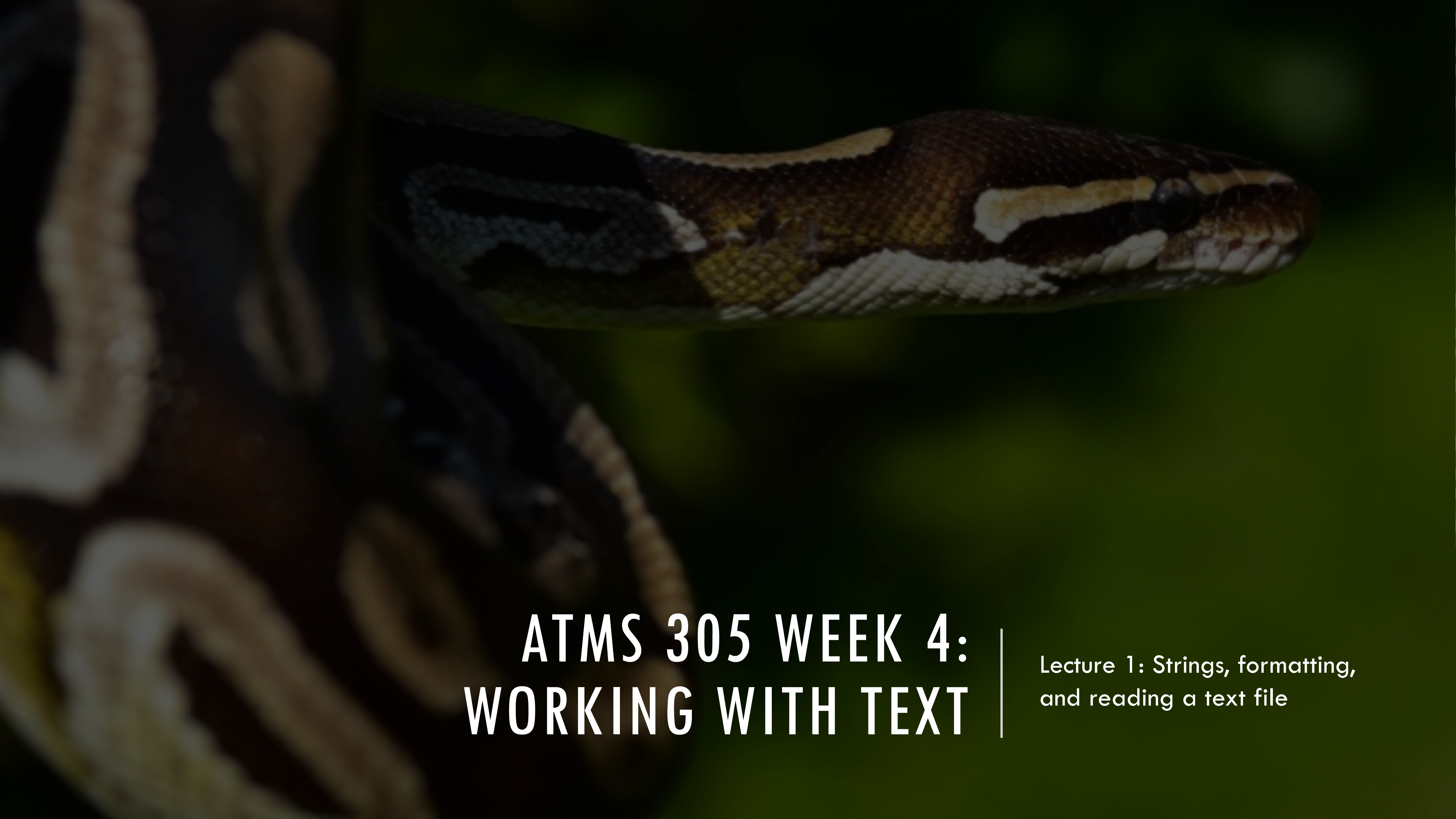

ATMS 305 WEEK 4: WORKING WITH TEXT

Lecture 1: Strings, formatting, and reading a text file

WHY SO HIGH STRUNG ABOUT STRINGS?

Strings are an important data format, so we will be spending time learning the ins and outs of using strings in python.

Luckily, strings are easy to work with in python.

Why are strings important?

- Important for software (filenames, user input and output)
- Data is often stored in strings, including in files (human readable = nice)
- Being able to turn strings into data is important (i.e., mining Twitter, translating text, reading text files)

STRINGS 101

In python, it is easy to define a string.

You can use single or double quotes:

```
message1 = 'hola'

message2 = "¿cómo estás?"  #special characters ok
```

String math for easy goodness:

```
message3 = message1+'!! '+message2

print(message3)
```

STRINGS IN LISTS AND TUPLES

We learned last week that strings can be part of a list or tuple.

```
suburbs =['Naperville','Gurnee','Evanston','Oak Park','Rosemount']
population = ('144864','31284','75520','52066','4236')
for sub, pop in zip(suburbs,population):
    print('The population of '+sub+', Illinois is '+pop+'.')
```

There are more elegant ways to do this, we'll cover this later.

ZIPPERING 🤐

zip allows you to put two lists together, like a zipper. The result is a list of tuples.

```
a = [1, 2, 3]
b = [4, 5, 6]
c = zip(a,b)
```

The asterisk (*) command, used with zip can unzip:

Try

```
d = zip(*c)
e = zip(*d)
```

GETTING ORGANIZED

Lists and tuples are great, but things can get disorganized if you have a lot of different variables.

How can we be more organized?

Answer: python **dictionaries**!

DICTIONARIES 📔

Dictionaries are similar to lists, except that the elements, rather than being indexed by integers, they are indexed by "keys". Keys may be numbers, strings, or objects. The contents of dictionaries can be strings, values, lists, etc.

3 ways for creating a dictionary:

Method 1: curly brackets

```
 d = {'quarterback': 'Brady', 'coach': 'Belichik', 
'wins':5}
```

keys: quarterback, coach, wins

values: Brady, Belichik, 5

DICTIONARIES 📒

Method 2: keywords

```
e = dict(first = 'salad', second = 'soup',
          third = 'main course')
```

Method 3: lists

```
f = dict([['country','Canada'],
          ['flag','maple leaf'],['population',3e7]])
```

ADDING ELEMENTS TO A DICTIONARY

```
e = dict(first = 'salad', second = 'soup',
          third = 'main course')
```

We can add elements to an existing dictionary by using the assignment operator, giving a new key and value pair

```
e['fourth'] = 'dessert'
```

You can also create an empty dictionary, and then add:

```
g = {}
g['fourth'] = 'Taco Bell'
```

Are dictionaries mutable or immutable?

FUNCTIONS AND METHODS FOR DICTIONARIES

`len(e)`	number of key/value pairs
`del e['third']`	delete a key/value pair
`k in e`	checks to see whether e contains an item with the key given by k returns True or False
`e.clear()`	clears a dictionary
`e.copy()`	returns the copy of a dictionary
`e.keys()`	returns the keys in a dictionary
`e.items()`	returns a list containing tuples of all of the key/value pairs
`e.values()`	returns the values in the dictionary

ADDING TO DICTIONARIES

Don’t get overly confused about adding dictionary keys versus adding elements to a list contained in a key.

For example, adding a key:

```
wxdata = {}

wxdata['maxtemp'] = [34,45,36]

wxdata['mintemp'] = [28,32,28]
```

is not the same as adding an entry to a list within a dictionary key:

```
wxdata['maxtemp'].append(43)

wxdata['mintemp'].append(31)
```

READING DATA FROM A FILE: THE HARD WAY

To get started processing strings, we will begin with something very useful, reading a text file into python.

We will start doing this at a low level using python's built in functions for reading data. We will practice using NOAA's global land and ocean temperature anomaly dataset. I downloaded the data from: https://www.ncdc.noaa.gov/data-access/marineocean-data/noaa-global-surface-temperature-noaaglobaltemp

The data files have annual average values of:

1st column = year 2nd column = anomaly of temperature (K) 3rd column = total error variance (K**2) 4th column = high-frequency error variance (K**2) 5th column = low-frequency error variance (K**2) 6th column = bias error variance (K**2)

READING A TEXT FILE IN PYTHON

The most basic way to read a text file is to use the open and read command.

```
filename='aravg.ann.land_ocean.90S.90N.v4.0.1.201612.asc'
with open(filename, "r") as f:
    alist = f.read().splitlines()
```

The r means read the file with read permissions only!
The with open() syntax opens and closes the file

What does `alist` look like?

What does splitlines() do? Try taking it off and see what happens.

SO I READ THE FILE, NOW WHAT

The result of the read command is a list, with each element a line in the file. That is not particularly useful if you want to look at the columns in the file.

To split the data, we can use the split() command to split the columns into separate items in a list:

Try

```
alist[0].split()
```

Each element is split by the space.

If the columns are not split with a space you can put an argument with that character in the split command:

```
alist[0].split[',']
```

ORGANIZING DATA

We could read all of the columns in the file and put them into a separate variable.

```
filename='aravg.ann.land_ocean.90S.90N.v4.0.1.201612.asc'
year = []
temperature = []
with open(filename, "r") as f:    #read all the lines in the file
    alist = f.read().splitlines()
for line in alist:  #iterate through lines
    year.append(line.split()[0])
    temperature.append(line.split()[1])
```

THAT'S GREAT, BUT WHAT IF WE HAVE LOTS OF VARIABLES?

Use a dictionary!

```
filename='aravg.ann.land_ocean.90S.90N.v4.0.1.201612.asc'
wxdata={'year':[],'temperature':[]}    #define dictionary
with open(filename, "r") as f:    #read all the lines in the file
    alist = f.read().splitlines()
for line in alist:  #iterate through lines
    wxdata['year'].append(line.split()[0])
    wxdata['temperature'].append(line.split()[1])
```

CONVERTING TO FLOATS

If we want to store the data as numbers, add the proper type conversion command:

```
filename='aravg.ann.land_ocean.90S.90N.v4.0.1.201612.asc'
wxdata={'year':[],'temperature':[]}    #define dictionary
with open(filename, "r") as f:    #read all the lines in the file
    alist = f.read().splitlines()
for line in alist:  #iterate through lines
    wxdata['year'].append(int(line.split()[0]))
    wxdata['temperature'].append(float(line.split()[1]))
```

QUERYING DATA

We can use several techniques to query data.

Lists can be queried using built in commands:

```
wxdata['year'].count(1986)
```

```
wxdata['temperature'][wxdata['year'].index(1986)]
```

Sort using a technique called **list comprehension.**

```
highesttemps = [i for i in wxdata['temperature'] if i >= 0.4]
```

```
highesttemps.sort()
```

Note that this just sorts the one column.

SORTING LISTS AND DICTIONARIES

Want to sort a list by another, or a dictionary by a column? We have the sort command.

```
X = ["a", "b", "c", "d", "e", "f", "g", "h", "i"]

Y = [ 0, 1, 1, 0, 1, 2, 2, 0, 1]

yx = zip(Y, X)

yx [(0, 'a'), (1, 'b'), (1, 'c'), (0, 'd'), (1, 'e'), (2, 'f'), (2, 'g')
, (0, 'h'), (1, 'i')]

yx.sort()

yx [(0, 'a'), (0, 'd'), (0, 'h'), (1, 'b'), (1, 'c'), (1, 'e'), (1, 'i')
, (2, 'f'), (2, 'g')]

x_sorted = [x for y, x in yx]

x_sorted

['a', 'd', 'h', 'b', 'c', 'e', 'i', 'f', 'g']
```

OR A ONE LINER

```
[x for y, x in sorted(zip(Y, X))]
```

How can we sort dictionaries in the same way? One way is to do the same procedure, reassigning a new dictionary.

Try it!

WHAT'D WE LEARN? 🤓

- **strings** are useful in programming and data analysis
- **zippering** can put multiple lists together for easy iteration
- **reading** a file is easy, need to parse the output
- **dictionaries** are flexible data structures that are defined by key/value pairs