

# Lecture 3

Pythons.

# PSAs

- Lectures may be changed at the last minute.
- Feedback (especially negative) is very much appreciated. Feel free to come to us with questions during the week.
- Start thinking about what other topics you'd like to cover.

# FAQ

- Terminal color change
  - Mac: Terminal -> Preferences->Text. The window that pops up allows you to create and save a custom scheme
  - Linux: Edit -> Profile -> New profile -> colors
- Casting

# Topics

- Review looping
- Input/output
  - How can you handle files with Python?

# Frequently Asked Questions

- Clearing a list:
  - Reassign it
  - `list = []`

# A slight digression: White space

- White space refers to the space between words and characters
  - In python, white space is generally not important
  - But there are two main things to be aware of:

# A slight digression: White space

- White space refers to the space between words and characters
  - In python, white space is generally not important
  - But there are two main things to be aware of:
- 1. Whitespace characters may be hidden in your text, but they're there
  - a. Common whitespace characters:  
`\t, \s, \n, \r`
- 2. Whitespace matters for indented code
  - a. As we've seen with loops

# Looping

- In its most basic form, the act of doing a task many times



# Looping

- In its most basic form, the act of doing a task many times
- Loops, along with other statements we'll cover, give your program *control flow*

# Input/Output

- You don't always want to type input into the terminal.

# Input/Output

- You don't always want to type input into the terminal.
- Instead, you might have a data file that you would like to open and use as input

# Input

- `open()` is one of the most common ways of doing this
  - `f = open('filename', 'mode')`
  - the 'filename' will be the file you want to open
  - 'mode' will be what you would like to do with this file
    - `r` for read will be assumed if no mode is provided
    - Read-only means you cannot write to the file
    - `w` will allow you to write to the file
    - `r+` will allow reading and writing

# Input example

- I have some data in a file. I'd like to open it, read it and write some lines to it, as well
  - `>>> f = open('locations.csv' , 'r+')`
  - `f` is now a file object
  - This simply opens the file in a way that will allow reading and writing

# Input

- Now what?
  - `>>> f.read()`
    - Returns your whole file as one big string. It will not be nicely formatted and will show whitespace characters.
  - `>>> f.readlines()`
    - I want you all to try this. Open the file in a text editor, and compare this to what you see on the screen

# Note

- Both of the previous commands read beginning to end-of-file
- Notice what happens if you run them sequentially
- `f.seek(0)`

# Input

- Now what?
- `f.readlines()`
  - This will create a list of all the lines in a file
  - Or, you can do a little looping
    - `>>> for line in f:`
    - `... print line`
  - Capture these to variables
    - `>>> myfile = file.read()`
    - `>>> location = file.readlines()`



# Input

```
>>> for lin in location:  
...     print(lin).split()
```

# Input

```
>>> for lin in location:  
...     print(lin).split()
```

- What has split done?
- What type of object is lin?

# Input

- We can manipulate `lin` as a string!

```
>>> for lin in location:  
...     print(lin).strip(',')
```

```
>>> loc_list = []
>>> for lin in location:
...     loc_list.append((lin).strip().split(','))

>>> for i in loc_list[0:]:
...     if len(i[0:]) == 4:
...         print 'looks good'
```

# Challenge One

- How could you modify our workflow so far to use tab-delimited data?
- We've provided a tab-delimited version of the same data for you to try this.
- Try building lists from different columns and rows in the matrix. Does the slicing behave like you'd expect?

Hint: What symbol does python expect for a tab? It's been in this lecture, but feel free to google.

# Parsing

- A very useful data structure is the dictionary
  - Like a real dictionary, this is a structure in which there is a key and a value
    - The key is a unique identifier by which you can call the variable.

```
>>> money_dict = {} # Dict initialize with {}
```

```
>>> for lin in loc_list:
```

```
...     money_dict[lin[0]]=lin[3]
```

# Parsing

```
>>> money_dict['Lake_Creek']
```

18

# Output

- Pretty similar to input!
  - But you need different permissions...
  - `>>> outfile = open('outfile.txt','w') #writing permission`
  - `>>> outfile.write(my_data_object)`
  - `>>> outfile.close()`



# Output

- `outfile = open('outfile.txt','w')`
- File modes:
  - 'w' will overwrite a file if it exists
    - Be careful with this! Make backups of important files often!
  - 'a' will append to a file
    - Your output will go at the end

# Output

- `outfile = open('outfile.txt','w')`
- File modes:
  - 'w' will overwrite a file if it exists
    - Be careful with this! Make backups of important files often!
  - 'a' will append to a file
    - Your output will go at the end

**With great power comes great responsibility!**

# Output

We can get fancy:

```
>>> outfile = open('out.txt', 'w')
>>> for item in money_dict.keys():
...     outfile.write('It cost %s dollars to sample %s location' %(money_dict
[item], item) + '\n')

>>> outfile.close()
```

# The with statement

- A 'with' statement calls an objects enter and exit methods
- Consider:

```
>>> with open('locations.csv') as f:  
...     data = f.read()
```

# The with statement

- A 'with' statement calls an objects enter and exit methods
- Consider:

```
>>> with open('locations.csv') as f:
```

```
...     data = f.read()
```

```
...
```

- If we type nothing else, this will execute read() and close the file for us. Easy!

# Comprehensions

- But to make full use of this wonderful statement, we should try a new way to create lists

# Comprehensions

- But to make full use of this wonderful statement, we should try a new way to create lists
- The **list comprehension** is a concise list constructor

# Comprehensions

- The paradigm so far:

```
for item in thing:
```

```
    list.append(item)
```



# Comprehensions

- The paradigm so far:

```
list = []
```

```
for item in thing:
```

```
    list.append(item)
```

- We can compact this

# Comprehensions

- We can compact this:

```
list = [item for item in thing]
```

We combine the initializing with the population of the list.

# Comprehensions

- We can compact this:

with open("locations.csv") as f:

```
loc_list = [line.strip().split(",") for line in f]
```

We combine the initialize the loc\_list

We populate the loc\_list with lines from f

We don't have to close the file - 'with' does this

# Exercise

- For either of the two provided files, or one of your own
  - Open the spreadsheet and read it in.
  - Choose a numerical column. Average it.
  - Write a statement about what mathematical operation you did, how you did it, and the result to a file

# Homework

- If you have a spreadsheet of your own data, think of two tasks you can do with that data. Try them. E-mail us the code you used, and the data. What worked? What did not work?
  - No Excel (yet)

# Homework

- If you don't have your own data, we have provided a set.

# Homework

- Read in the data
- Try
  - Checking for missing values

# Homework

- Read in the data
- Try some data quality control
  - Checking for missing values
  - Check that each column has the right data type

Column One	Strings
Column Two	Numbers
Column Tree	Numbers
Column Four	Numbers, all of which are unique



# Homework

- Read in the data
- Try some data quality control
  - Checking for missing values
  - Check that each column has the right data type

Column One	Strings
Column Two	Numbers
Column Tree	Numbers
Column Four	Numbers, all of which are unique

# Hints

Checking for missing values

How many values should be in each row? How can we check this? Subjective: What should we do with missing values? This is a real issue in almost everyone's work!

Check that each column has the right data type

This is a hard one. Think carefully about how to isolate data column-wise.

If a string is a number, what must it be possible to cast it as?

The last column is an extra special challenge. How might the set data type help with this?: <http://docs.python.org/2/library/stdtypes.html#set>