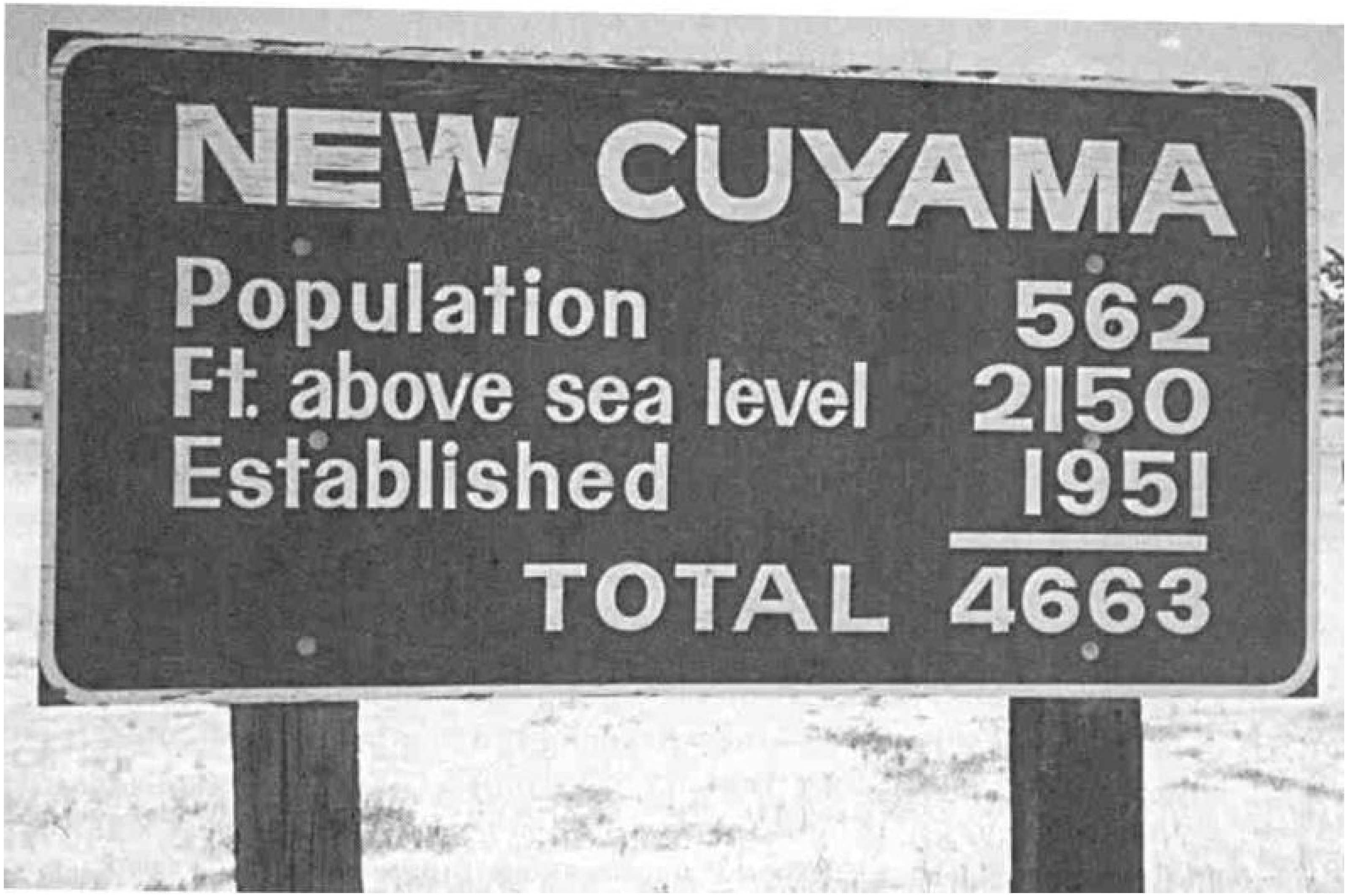
Lecture 2 - Data Bootcamp

Getting the goods on understanding data



- everything up until pandas we saw in 242, but now we will go a little deeper
- obviously we don't have enough time to get into everthing in this class, but we will try to nerd out a little today

Basic Data Types and Operations

- what is a variable?
    - a variable is a location in memory
    - python takes care of this for us, on-the-fly and in a way that we never actually see
        - but if we take a peak beind the curtain, we can actually see how it actually works

In [ ]:
```
x = 5
print (str(x)+ " and this variable's physical memory address is "+str(id(x)))
```

- now, this is a good time to talk about *mutability*
    - mutability mean that variables can be changed without being erased.
    - some types of variables like integers are *immutable*
        - this means if we change them, they get a different memory address, and the old memory address gets recycled (thanks python!)

In [ ]:

```
y = 5
print (str(y)+ " and this variable's physical memory address is "+str(id(y)))

y = "hello"
print (y+ " and this variable's physical memory address is "+str(id(y)))
```

mutability continued

- other variables can be changed in place, these are the mutable types
    - these are more complex data types, usually lists of information, dictionaries, data objects, and data frames

In [ ]:

```
l = ["Hello","Goodbye"] # this is an array

print (str(l)+ " and this variables physical memory address is "+str(id(l)))

l.append("Coriander") # add this string to the array!

print (str(l)+ " and this variables physical memory address is "+str(id(l)))
```

- mutability is important because of how memory management works
    - you can think about it in terms of the blockchain (Dodgecoin)
    - you don't want things to get overwritten
    - python does this for us without us seeing, which is great. in other languages you have to do it yourself!
        - if you don't use memory management well, you create 'leaks'. this causes the computer to crash!
        - in C you would have things called 'pointers' that are variables that track where memory exists. However, it used to be easy to get it wrong, and you could overwrite memory that is crtically important (like the memory that stores where the operating system exists!)
    - for us we only need to think about memory if we load data
    - lets say you have an array. then you add to it

In [ ]:

```python
import sys # a systems command library

myArray = ["hello friends"] # simple array with one item
i =0 # simple integer counter

# to make this more understandable
byt = sys.getsizeof(myArray)
kb = byt/1024 #kilobytes
mb = kb/1024 #megabytes

# how much memory does this array take up?
print("myArray as bytes:"+str(byt)+"\n\tmyArray as kb:"+str(kb)+"\n\tmyArray as mb:"+str
(mb))
```

In [ ]:

```python
# lets add some content to this array...
i
while i < 1:
    myArray.append("adding item: "+str(i))
# what could go wrong ?

# use stop button after a bit
```

```
In [ ]: print(str(len(myArray))) # print out the number of items in the array
```

```
In [ ]: import sys # we don't need to do this again, but in case this cell gets run out of seque
        nce....

        # to make this more understandable
        byt = sys.getsizeof(myArray)
        kb = byt/1024
        mb = kb/1024

        print("myArray as bytes:"+str(byt)+"\n\tmyArray as kb:"+str(kb)+"\n\tmyArray as mb:"+str
        (mb))
```

- in sum. memory management matters
- mutable variables means we have to be careful about how we load and store data in variables

but wait, what kinds of variables are there ?

simple variable types

basic data types and binary storage

- int (1,2,3,4)
- float (1.1,2.2,3.4)
- bool
- char/string "Hello World"

int

- integer variables are whole numbers

In [ ]:
```
# here is an int
x = 5
print("type of x:"+type(x).__name__)
print("size of x:"+str(sys.getsizeof(x)))
```

float

- floating point numbers are numbers with decimal values

```
In [ ]: # here is a float
        y = 555555.533333
        print("type of x:"+type(y).__name__)
        print("size of x:"+str(sys.getsizeof(y)))
```

ints are whole numbers. floats are decimal. but what happens when they mix?

- when two types of variables mix, it is called casting
    - some types of casting are implicit.
        - this means that python seeing two type of data and makes a decision on how to match them for us

In [ ]:
```
print(str(x/3))
```

In [ ]:
```
y = x/3 #implicit casting
type(y)
```

In [ ]:
```
y = int(x/3) #explicit casting
type (y)
# y
```

Bool

- boolean variables are only True or false
- they are used for conditional statements and flags
    - last week we used a boolean flag to set if out cake was done in our pseduocode

In [ ]:
```
x = True
```

In [ ]:
```
sys.getsizeof(x)
# funny, its about the size of an int....
```

can you cast a boolean?

```
In [ ]: int(x)
```

```
In [ ]: y = False
        int(y)
```

what are bool good for?

- booleans allow us to test logic
- booleans give us the power of conditionals

In [ ]:
```
x = 4
y = 5
x < 5
```

type of conditionals

- there are many forms of conditionals
    - Equal to (==)
    - NOT (!)
    - OR (|) - inclusive
    - AND (&) -exclusive
    - exclusive OR (^) very exclusive. only if opposite
    - Greater Than (>)
    - Less Than (<)

```
In [ ]: # equal to
        x = 5
        y = 4
        x == y
```

```
In [ ]: print(True == False)
        print(True == True)
        print(False == False)
```

In [ ]:
```
# not
x = 5
y = 4
print(x != y)
```

In [ ]:
```
print(True != False)
print(True != True)
```

In [ ]:
```
# or
x = True
y = False

print(x | y)
```

In [ ]:
```
print(True | False)
print(True | True)
print(False | False)
```

In [ ]:
```
# and
x & y
```

In [ ]:
```
print(True ^ True)
print(False ^ False)
print(True ^ False)
print(False ^ True)
```

In [ ]:
```
#greater than
x > y
```

In [ ]:
```
#less than
x < y
```

char and strings

- generally, chars (characters) are single items 'a' or 'b'
- strings are sets of characters 'h','e','l','l','o'
- in python, all chars are strings.

In [ ]:
```
c = 'h'
type(c)
```

In [ ]:
```
c = 'hello'
type(c)
```

In [ ]:
```
sys.getsizeof(c)
```

strings have cool properties and methods

- for example, we can use strings like they are arrays (more in these in a sec)

In [ ]:
```
#here is a protery of the string c
c = 'hello'
c[3]
```

In [ ]:
```
# we could also add two strings together
strA = "Hello"
strB = " Michael"
strA+strB
```

In [ ]:
```
# here is a method we can call upon for var (variable) c
c.upper()

#strings have lots of method/functions we can call
# for more of these, check out:
# c.capitalize()
# c.count('l')
```

lastly for now, escape sequences

- escape sequences are how we are able to print out spceial items in a string
- for example, how can you tell python to print items on a new line? how can you use a tab?
- escape sequences are string modifiers. the usually begin with a "\" and then a letter
    - a new line is \n and a tab is \t

In [ ]:
```
strA = 'hello \n how are you'
print(strA)
```

In [ ]:
```
strB = "table layout \n1 \t2 \t3\n4 \t5 \t6\n7\t8\t9"
print(strB)
```

but how do i print out an escape sequence?

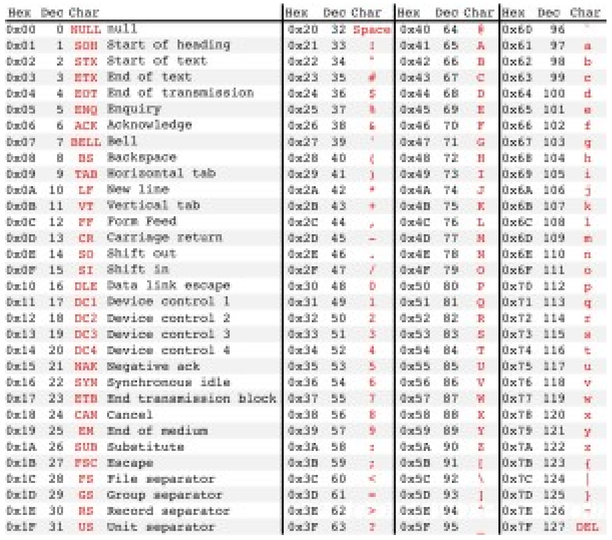
- add another "\"

```
In [ ]:
strC = "escape sequences are cool the the new line (\\n) and tab (\\t) escape \n\t see?
\n\t\t...\n\t\t\t..."
print(strC)
```

string casting

- can you store the letter "s" on the hard disk?
- can you store the number 5 on the hard disk?
- if you can't store these letters and numbers, how are they stored?

- strings are 'encoded' into memory
    - and different era's of computing used different kinds of encoding.
    - early encoding was ASCII
        - ascii gave us 256 values (8 bytes) to work with



Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	ESC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

there are different types of encoding out there

- ascii worked for a while
- see:ascii art

```
         :LOL:ROFL:ROFL
              /\
L   /___________
O ===          []\
L     \            \
       \__________  ]
          I    I
       ---I----I----/

      ROFL COPTER!!!
```

- courtesy wikipedia: [https://en.wikipedia.org/wiki/ASCII_art (https://en.wikipedia.org/wiki/ASCII_art)](https://en.wikipedia.org/wiki/ASCII_art)

clearly 256 characters isn't going to cut it.

- more often now, we use unicode
    - unicode has multiple types of encoding for character sets (english and arabic for example)
    - unicode increases the size of characters from 256 to 143,859
        - see:https://en.wikipedia.org/wiki/Unicode (https://en.wikipedia.org/wiki/Unicode)
    - this matters because if you try display a string in one encoding type and it is another, you will have trouble

In [ ]:
```
# let s be a string we want to convert to unicode character
s = "\U0001F590".encode("UTF-8")
print(s)
```

In [ ]:
```
# okay, now decode the value again
print(s.decode())
```

Arrays

- arrays are lists of data
- we have already seen them a couple times
    - we saw them in 242 last year of course
    - but we also saw them in lecture 1 with our names example, and today with strings
- arrays are a fantastically lightweight way to store information

In [ ]:

```
names = ["Anderson, Matthew","Barclay, Amy","Franklyn, Josh","Hong, Henry",
         "Jodinata, Dominicus","Jovner, Georgi","Kolston, Sophie","Lee, Jonathon",
         "Lee, Justin","Lewis, Cody","Lim-Yip, Alyssandra","Lin, Xizhe","Machado da",
         "Cruz, Jenna","McKay, Ben","Mead, Alexander","Milliken, Thomas","Mukherjee, Sha
lini",
         "Narayan, Kaveesh","Parsons, Thomas","Patel, Dex","Pedersen, Danielle","Richard
son, Joshua",
         "Sakik, Saima","Smith, Bella","Tenedero, Julia","Ware, Charli","Webster, Lucy"]
names
```

- arrays are indexed
    - the index starts at 0
    - for the array ["hello","hi","gidday"]
        - hello is index 0
        - hi is index 1
        - gidday is index 2
    - but the length of the array is 3
    - indexes are denoted by the [] characters

In [ ]:
```
arr = ["hello","hi","gidday"]
arr[0]
```

In [ ]:
```
len(arr)
```

In [ ]:
```
arr[3]
```

arrays are lists of anything. including lists

- this is where arrays really get useful
- arrays are just collections of variables
- you can mix variable types in arrays. I don't recommend it.

In [ ]:
```
arr = ['hello',3,True]
sum(arr)
```

In [ ]:
```
arr = [3,4,5]
sum(arr)
```

arrays can have arrays as items

- this is called a multi dimensional array

In [ ]:
```
arr
arr2= [1,2,arr]
arr2
```

- a better idea is to make your dimensions consistent
    - this way you know you can expect elements in specific places. later in the course we are likely to have pretty crazy arrays, this will help a lot!

In [ ]:
```
arr = [[1,2,3],[4,5,6],[7,8,9]]
arr
```

In [ ]:
```
#is the same as
arr = [[1,2,3],
       [4,5,6],
       [7,8,9]]
```

arrays have a couple interesting functions you'll use regularly

- len() is the number of items in an array
- append() adds a new item to the array

Break!

patterns. if and loops

- if, for, and while
    - these basic patterns get used over and over and over in programming. they allow us to process data relative to what we epect to see, and what we don't expect to see
    - introducing if, for, and while introduces us to encapulation and tab-indentation
        - anything that is tab-indented in something else belongs to it.

- IF I GO TO THE STORE
    - THEN I SHOULD DO THIS THING
- OTHERWISE
    - DO THIS OTHER THING
    - OH AND THIS OTHER THING TOO

if

- if is what changes the program flow based on data
- there are three types of statements that we can use with if statements
    - if, else if,else
- if statements are where boolean logic starts to get useful
    - if the logic is true, do this
    - else if, the logic is slightly different and true, do this
    - else, well fall back to this final thing

In [ ]:
```
# if
x = 1
y = 2

if (x < y):
    print(str(x)+" is less than "+str(y))
```

In [ ]:
```python
# if-else
name = "Michael Martin"

if (name == "Michael Martin"):
    print("you are the teaching leading this lecture")
else:
    print ("you could be a teacher or a student")
```

In [ ]:
```python
# if-elif-else
name = "Michael Martin"
# name = "Sila"
# name = "some other name"

if (name == "Michael Martin"):
    print("you are the teacher leading this lecture")
elif (name == "Sila"):
    print ("ah, ok, you are also a teacher")
else:
    print("you are not Michael or Sila, you must be a student")
```

for loop

- in python, for loops allow us to iterate through 'iterable' items
    - yes that is confusing, but it really means anything that has sub-items in it
    - an array is an iterable item

In [ ]:
```
# for loop
arr = [1,2,3]

# for an item 'i' in the array 'arr'
for i in arr:
    print ("item:"+str(i)) # print out the current item
```

- the for loop is deceptively useful.
    - you could use it to go through every polygon in a shapefile, for example

while loop

- while loops are like if statements, but far more dangerous.
    - remember at the beignning of this class when we intentionally made a memory leak?
    - imagine that, but you have no idea how you did it
    - that is what while loops do.

In [ ]:

```
# while loops evaluate a condition at the top of every iteration
# they continue until that condition evaluates to false
while (true):
    print("this loop will never end")
```

- while loops are good if you have a non-iterable set of items that you need to get through
- for example, you just want your array to count to 10

In [ ]:
```
# a simple loop to count to 10

i = 0 # this is called a counter
while (i<10): # keep going until i > 10
    print("i equals:"+str(i))
    i = i+1 # increment counter by 1

# what if we wanted it to actually go until it print i=10?
```

Moving on to more complex data types

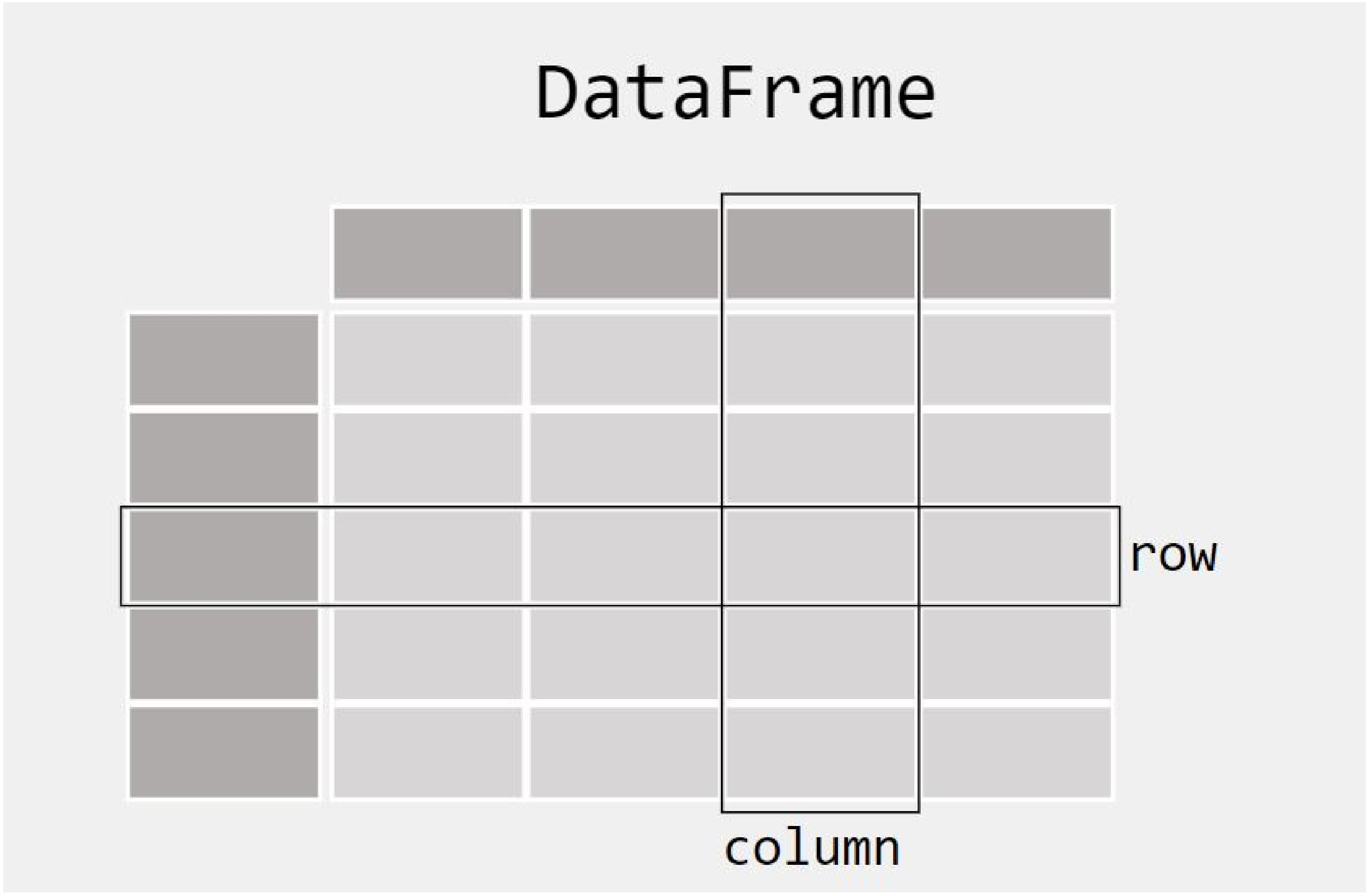
- one last thing. Pandas.

Pandas

- for real though, what is pandas?
    - pandas is a *very* powerful data handling and processing library for python.
    - it has a blazing fast ability to load and save data from a wide variety of formats (csv,json,excel, etc)
    - it can transform data very quickly, too.
- ok, how do I get it?
    - run command prompt from anaconda
        - if you forget how, have a look at last week's lecture :)
    - type: pip install pandas

Introducing, the dataframe

- pandas is all organized around a concept called a DataFrame
- the dataframe is like a crazzzzzzzy powerful 2D array
- pandas brings data transformation, statistical analys, and plotting/visualization directly to you



In [ ]:
```python
# to get started, we need to import the pandas library
# if you get lost in class today, I highly recommend the pandas website
# the tutorials on the site are excellent!
# https://pandas.pydata.org/docs/getting_started
import pandas as pd

# to make a dataframe we can easily construct one ourselves
#creating a DataFrame using a dictionary
# we haven't covered dictionaries in detail yet, but we will be next week
# for now, all you need to know is that a dictionary is a
# type of data strcutre that stores information as "key":"value" pairs
# and like arrays the "value" can be any tpye of data, like in this case, an array
df = pd.DataFrame(
    {
        "Name":[
            "Braund, Mr. Owen Harris",
            "Allen, Mr. William Henry",
            "Bonnell, Miss. Elizabeth",
        ],
        "Age": [22, 35, 58],
        "Sex": ["male", "male", "female"],
    })
df
```

query the table, asking for a single column of information

- in pandas, a column is a 'series'

In [ ]:

```
# get a series (column) of data
df["Age"]
```

pandas make summarizing data easy

```
In [ ]:
# we can also simply as pandas for stats
# https://pandas.pydata.org/docs/getting_started/intro_tutorials/06_calculate_statistic
s.html
df.describe()
```

pandas does plot the easy way!

```
In [ ]: # we can also ask pandas for a really simple graph of the data
        # https://pandas.pydata.org/docs/getting_started/intro_tutorials/04_plotting.html
        df.plot.bar(x='Name',y='Age')

        # there are lots of different options for the type and styling of plots, far too much fo
        r today!
        # for example these are all the types of plots!
        #   'area','bar','barh','box','density',
        #   'hexbin', 'hist', 'kde', 'line', 'pie', 'scatter'
```

lets make this interesting

- over the weekend I got interested in harvesting Reddit.com data
    - reddit is an 'open api' meaning all the publically posted information is free to grab and play with
    - I'm going to gloss over how praw works a little over the next couple slides, but suffice to say it is pretty slick and makes getting data easy
    - if you want to install praw, just use
        - pip install praw

First I’m going to collect login information

- In the background I've created a developer account for reddit
    - I've written a file called reddit_app_login.json to sotre my login info
    - This is really important, as I don't want to broadcast my details with the code
    - A blank version is given with this week's zip file
    - you can create an account usingthe links in the code below

In [ ]:

```
# useful links
# https://medium.com/@plog397/webscraping-reddit-python-reddit-api-wrapper-praw-tutorial
-for-windows-a9106397d75e
# https://praw.readthedocs.io/en/latest/tutorials/comments.html
import praw # import the main module
from praw.models import MoreComments # a set of classes to handle comments
import datetime as dt # handling some time formats


# get account information from a JSON file
import json
with open("reddit_app_login.json","r") as read_file:
    rKeys = json.load(read_file)
# rKeys.keys() # check to see if the right keys were collected from the JSON file
```

getting the data from reddit

- this is called 'calling' the API
- we use this code to 'ask' reddit for the data we want

In [ ]:

```
#reddit account information
reddit = praw.Reddit(
    client_id=rKeys["client_id"],
    client_secret=rKeys["client_secret"],
    password=rKeys["password"],
    user_agent=rKeys["user_agent"],
    username=rKeys["username"],
)

# We are ready to start grabbing data from reddit
# lets grab from r/auckland
top_n = 1000 # how many posts would you like to get ?
r_auckland = reddit.subreddit('auckland')
new_auckland = r_auckland.hot(limit=top_n) # this time lets use the 'hot' posts

# Verify that we are recieving data
# this is a large amount of data so probably not going to do this.
# for i in new_auckland:
#     print(i.title)
```

getting dataframe ready

- once we have the information, it isn't nessecarilly ready to be loaded directly in to a dataframe
- its a simple matter of extracting the data in the new_auckland variable into a format pandas understands
    - in this case it is a dictionary

In [ ]:

```
# Now that we have the data from praw we want to format from its raw data
# into something we can stick into a Pandas Dataframe

# we make an empty dictionary called sub_post that can hold all data from the subreddit
sub_posts = {
    "title":[],
    "subreddit":[],
    "score":[],
    "id":[],
    "url":[],
    "comms_num": [],
    "created_timestamp": [],
    "created_datetime":[],
    "body":[]}

# now, for each of the items in the subreddit that we found we are going
# to add the data from it into the dictionary keys as array items
for j in new_auckland:
    sub_posts["title"].append(j.title) # note the append method. it is the same as we sa
w before for arrays
    sub_posts['subreddit'].append(j.subreddit)
    sub_posts["score"].append(j.score)
    sub_posts["id"].append(j.id)
    sub_posts["url"].append(j.url)
    sub_posts["comms_num"].append(j.num_comments)
    sub_posts["created_timestamp"].append(j.created)
    sub_posts["created_datetime"].append(dt.datetime.fromtimestamp(j.created))
    sub_posts["body"].append(j.selftext)


# now that the dictioary is in the shape pandas wants we
# convert our data into a pandas dataframe
rAuckland_df = pd.DataFrame(sub_posts)

# and now, lets write this dataframe to file
# in case anything happens to the data we can just reload it
```

What did we just do?

- if you look at the folder where this notbook is located, there will now be a file 'top1000_rAuckland_posts.csv'

```
In [ ]: # lets take a look at what pandas sees
        # by defualt, pandas loads the top and bottom 10 items when we ask for information
        rAuckland_df
```

In [ ]:

```
# what are the stats of this?
rAuckland_df.describe()
```

we can go deeper

- getting all the posts was cool, but waht would be really cool would be to see all the comments posted in the posts
- lets try and get every top level comment for every post ever made in r/Auckland

In [ ]:
```
# lets now put everything together into a master table of all the top posts from all the
# top 1000 hot posts on the subreddit

# create a dictionary object that will house all of our data for the time being,
# until we load it all into the Pandas Dataframe
# same as before
top_comments = {
    "post_title":[],
    "subreddit_name":[],
    "subreddit_id":[],
    "post_id":[],
    "comment_id":[],
    "comment_author":[],
    "comment_body":[],
    "created_utc":[],
    "created_datetime":[],
    "permalink":[],
    "comment_score":[],
    }
```

two layer deep loops

- remember how we can have an array inside an array?
- well we can have a loop inside a loop!
    - here we are just getting all the comments, for every post and adding it to a big master list
- the code below has all been commented out to remind me not to run it.
    - it all works, it just takes a while and I don't want to waste out time waiting for all the comments to download

```
In [ ]: # OK This one is going to be a bit of a doozey, but its not that hard really. we just ne
        ed to follow the logic
        # 1. for each of the top posts that we have gathered in the preceding dataframe
        # 2. go through each of the top comments in the post
        # 3. store the data in that top post as new data in the top_comments dictionary

        # first, for each id in the rAuckland DataFrame
        # for p in rAuckland_df["id"]:
        #     current_post = reddit.submission(id=p) # get the data from the submission/post

        #     # now, lets loop through the top level comments in the submission/post
        #     for top_level_comment in current_post.comments:
        #         if isinstance(top_level_comment, MoreComments): # this is a little helper to m
        ake sure data exists
        #             continue
        #         top_comments["post_title"].append(top_level_comment.submission.title)
        #         top_comments["subreddit_name"].append(top_level_comment.subreddit.display_nam
        e)
        #         top_comments["subreddit_id"].append(top_level_comment.subreddit_id)
        #         top_comments["post_id"].append(top_level_comment.parent_id)
        #         top_comments["comment_id"].append(top_level_comment.id)
        #         top_comments["comment_author"].append(top_level_comment.author)
        #         top_comments["comment_body"].append(top_level_comment.body)
        #         top_comments["created_utc"].append(top_level_comment.created_utc)
        #         top_comments["created_datetime"].append(dt.datetime.fromtimestamp(top_level_co
        mment.created_utc))
        #         top_comments["permalink"].append(top_level_comment.permalink)
        #         top_comments["comment_score"].append(top_level_comment.score)

        # # # And thats it! We have amde a master list of all the top comments for the top n pos
        ts of the reddit of interest!
        # # create a pandas dataframe from the dictionary
        # reddit_df = pd.DataFrame (top_comments)

        # # and save for later
        # filename="rAuckland_top_comments.csv"
        # reddit_df.to_csv(filename)
```

Thats going to take a while...

- actually its pretty fast given how much data its asking for
- but in the meantime we can load the data that came with this folder for the lecture that I created ahead of time

In [ ]:
```
# lets explore this dataset a little...

# load reddit_df from the csv we created.
filename="old_rAuckland_top_comments.csv"
reddit_df = pd.read_csv(filename,parse_dates=["created_datetime"]) # read the file and p
arse the datetime test as pandas datetime objets

reddit_df
```

thats a lot of data. also a lot of posts with few comments.

- lets investigate!

In [ ]:
```
reddit_df["comment_score"].describe()
```

In [ ]:
```
reddit_df["comment_score"].plot()
```

investigation. When is the best time to post for max comments?

```
In [ ]: # https://pandas.pydata.org/docs/getting_started/intro_tutorials/09_timeseries.html
        import matplotlib.pyplot as plt
        fig,axs = plt.subplots(figsize=(12,4))
        reddit_df.groupby(reddit_df["created_datetime"].dt.hour)["comment_score"].max().plot(kin
        d='bar',rot=0,ax=axs)
        plt.xlabel("hour of the day");
        plt.ylabel("avg comment score");
```

does comment length correlate to karma?

```
In [ ]:
# https://pandas.pydata.org/docs/getting_started/intro_tutorials/10_text_data.html
reddit_df["comment_length"] = reddit_df["comment_body"].str.len()
reddit_df.plot.scatter(y="comment_score",x="comment_length",alpha=0.5,figsize=(12,6),log
x=True) # Check out the axis!
```

investigation. Who has the most single comment karma from r/auckland?

- to answer this question we have to do something quite algorithmically taxing, groupby
    - to group all of the posts that are related to specific users, we have to sort, then summarize
    - in this groupby clause we use the .max() method that sums as it goes. Its kind of like the summary statistic you can add to a spatial join function in ArcGIS (*hiss*)

In [ ]:
```
fig,axs = plt.subplots(figsize=(12,4))
# in this clause we use groupby. its a taxing operation
reddit_df.groupby(reddit_df["comment_author"])["comment_score"].max().plot(kind='bar',ro
t=0,ax=axs)
plt.xlabel("comment author");
plt.ylabel("avg comment score");
```

final investigation. What user has the highest karma, from r/Auckland?

```
In [ ]:
# create a new series 'k' by grouping name with summed karma
k = reddit_df.groupby(reddit_df["comment_author"])["comment_score"].sum()
#convert the series to a dataframe
l = pd.DataFrame(k)
# output a sorted version, and make permanent
l.sort_values(by=["comment_score"],ascending=False,inplace=True)

# the default version of this is too big a table, so lets just grab
# the top and bottom 5
overall_karma = pd.concat([l.head(5),l.tail(5)])
overall_karma
```

```
In [ ]: overall_karma.plot()
```

Group Activity

- in your groups, go through the /r/wallstreetbets dataset and see what you can come up with!
  - I've put a dataset in this week's zip file on Canvas called 'reddit_wsb.csv'
  - the data comes from Kaggle.com
- Using we just looked at, plus using the pandas website, what can you hack together right now?
  - some ideas to get you started:
    - who posted the most?
    - when was the best time to get karma?

In [ ]:
```python
# a dataset from r/wallstreetbets
# collected on kaggle using python lib praw
# dataset url: https://www.kaggle.com/gpreda/reddit-wallstreetsbets-posts

# here is some code to get you started
import pandas as pd
wsb = pd.read_csv("reddit_wsb.csv")
wsb.describe()
```

In [ ]:
```python

```