Coursera: Python for Everybody Specialization

A How-to-Guide

Alex Huerta

BUS 464-12: Barry D. Floyd

4 Sep. 2019

The specialization is broken down into 5 courses, each of them containing weeks of information/lessons, quizzes, and assignments. The Python for Everybody Specialization course layout is detailed as follows:

- Course 1: Programming for Everybody
  - Week 1 - Ch.1: Why We Program
  - Week 2 - Installing and Using Python
  - Week 3 - Ch.1: Why We Program (cont.)
  - Week 4 - Ch.2: Variables and Expressions
  - Week 5 - Ch.3: Conditional Code
  - Week 6 - Ch.4: Functions
  - Week 7 - Ch.5: Loops and Iteration
- Course 2: Python Data Structures
  - Week 1 - Ch.6: Strings
  - Week 2 - Unit: Installing and Using Python
  - Week 3 - Ch.7: Files
  - Week 4 - Ch.8: Lists
  - Week 5 - Ch.9: Dictionaries
  - Week 6 - Ch.10: Tuples
  - Week 7 - Graduation
- Course 3: Using Python to Access Web Data
  - Week 1 - Getting Started
  - Week 2 - Ch.11: Regular Expressions
  - Week 3 - Ch.12: Networks and Sockets
  - Week 4 - Ch.12: Programs that Surf the Web
  - Week 5 - Ch.13: Web Services and XML
  - Week 6 - Ch.13: JSON and REST Architecture
- Course 4: Using Databases with Python
  - Week 1 - Object Oriented Python
  - Week 2 - Basic Structured Query Language
  - Week 3 - Data Models and Relational SQL
  - Week 4 - Many-to-Many Relationships in SQL
  - Week 5 - Databases and Visualization
- Course 5: Capstone: Retrieving, Processing and Visualizing data with Python
  - Week 1 - Welcome to the Capstone
  - Week 2 - Building a Search Engine
  - Week 3 - Exploring Data Sources (Project)
  - Week 4 - Spidering and Modeling Email Data
  - Week 5 - Accessing New Data Sources (Project)

Week 6 - Visualizing Email Data
Week 7 - Visualizing New Data Sources (Project)

Recommendations

While this may not be true for every one, one of the reasons why I was able to grasp the knowledge in the python courses without too much difficulty was because I had already learned how to code in Java prior to taking the courses. Learning Java before taking Python helped immensely. Both of the languages are similar; so a lot of the key concepts are the same such as: strings, variables, classes, objects, methods, inheritance, arrays, if statements, loops, web scraping, etc. I will admit however, that Java is a little harder to learn than Python. Syntactically, Java is way more formal and requires extra code for similar commands in python, like printing, getting user input, declaring strings & variables, etc. For example, take the the print statement. In Java it's "system.out.print("x")", whereas in python it's just "print("x")". By learning the harder syntax first, it makes learning python much easier as it feels simpler to use and more straight forward in a sense.

Around the time I got to Course 3, I also started learning SQL, which came in handy as Course 4 is all SQL based. While SQL and Python are significantly different, by knowing both you are able to accomplish so much like: pull information from the web, read it, build a database, store the database with the info pulled from the internet and then visualize it using javascript. Overall, because of Python's simplicity and compatibility, the more programming languages you know, the better.

Another option that aids in ensuring success in the Python course is to watch a beginners tutorial ahead of time. Watching a tutorial will give you a brief overlook of the whole language, making it familiar and easier to understand when going through the course. Personally, I watched a four and a half hour Python tutorial offered by freecodecamp.org on YouTube (https://www.youtube.com/watch?v=rfscVS0vtbw). The tutorial recommended to download Python 3 instead of version 2, as it's the newer supported version, and more organizations are using version 3. Another recommendation was to use the IDE Pycharm for coding, which I personally like. Pycharm is easy to use and highlights certain terms to make the code easier to read. The tutorial covers the following topics:

Installing Python, Printing to the Screen (Hello World, Drawing Shapes), Variables & Data types, Working with Strings & Numbers, Getting input from users, Building a Basic Calculator, Mad Libs Game, Lists, List functions, Tuples, Functions, Return Statements, If Statements and Comparisons, Building a Better Calculator, Dictionaries, Loops, Build a Guessing Game, Exponent Functions, 2D

Lists & Nested Loops, Building a Translator, Try/Except, Reading/Writing Files, Modules & Pip, Classes & Objects, Building a Multiple Choice Quiz, Object Functions, and the Python Interpreter.

Watching the tutorial gave me a glimpse of what to expect and actually helped me speed through some of the early sections in the courses.

Another time saving recommendation, if you plan on applying for financial aid, you must fill out an individual application for each of the five courses. Filling out the application under the Python for Everybody Specialization (all 5 courses) will only submit an application for the first course. It takes two weeks for Coursera to review financial aid applications, so if you wish to avoid any unwanted time delays, ensure that you submit an application for each course.

Plan your time accordingly, the time estimates that Coursera gives doesn't factor in the extra time it will take to pause and repeat videos. In my case, it usually took me double the time that the time estimates predicted to get past the material. Lastly, as it will be thoroughly mentioned throughout the courses, it is very beneficial to download the "Sample code" folder from https://www.py4e.com/materials and the lecture slides from https://www.py4e.com/lessons, so you can follow along.

Course 1: Programming for Everybody (Getting Started with Python)

Course 1 is broken down into 7 weeks, has 5 quizzes, and 7 assignments. If you happened to watch the Python tutorial on YouTube prior to starting the course, you should be able to breeze through it fairly quickly. All of the topics were covered in the tutorial, and as it's the first course in the specialization, it only covers the easier introductory topics. Week 1, "Ch.1: Why We Program?," is all about welcoming you to the course. There's no coding to be done, quizzes or assignments, just videos and instructions on Python 3, how to download course texts, submit assignments, and a hardware/software overview (CPU, RAM, input/output devices, Main & secondary memory). Week 2, "Installing and Using Python", is very similar to week 1, there's no quizzes or assignments, and there's only one line of code you learn. For the most part week 2 shows you how to actually install python and then write your first program, which is just printing out "hello world" to the screen, and how to use the "Python Playground", the interface used to submit coding assignments.

Week 3, "Ch.1: Why We Program (cont.)," is where the course starts lecturing more in-depth regarding python coding, so you want to make sure that Python has been installed on your computer. There's a lecture video that gives an overview of writing paragraphs of code, the logic of how the code is read/ran, and gives peaks of the structure one's code needs follow for topics that will be learned later like: if statements,

loops, dictionaries, etc. One major takeaway is that Python reades code, top-down. In addition there's a Quiz on the Ch.1 Material (Terms, and basic logic of code structure), as well as the first assignment, "Hello World." See appendix B.1 for more information.

Week 4, "Ch.2: Variables and Expression," is pretty self-descriptive as you watch three lecture videos on variables, expressions, constants, reserved words, type, conversion between types, operators, operator precedence, user input, integer division, and comments. One of the key concepts to understanding week 4 is to think of variables as containers for storing data and expressions as the ways to represent that data. There's a quiz on the concepts described, as well as two assignments that require utilization of those concepts. See appendix B.2 for more information.

Week 5, "Ch.3: Conditional Code," explains logic based coding by introducing topics such as: If, Else, and Elif statements, which are used to make one-way, two-way, or multi-way decisions with certain conditions. Some of the others topics to focus on are the comparison operators: == (equal to), <=, >=, <, >, != (not equal), and the "try/except" which is used to catch errors. You will then be quizzed on the topics and assigned two assignments to put them in practice. See appendix B.3 for more information.

Week 6, "Ch.4: Functions," is pretty straight-forward and covers functions, which are easiest to think of as reusable pieces of code or "stored" code that we use. They are very useful for saving time and code. The important thing to remember is that functions take some input (parameters) and processes an output (return value). It's important to pay attention to the lectures on how to build and use functions as you'll be quizzed on them and be given an assignment on them. See appendix B.4 for more information.

Week 7, "Ch.5: Loops," covers everything regarding loops. You can think of loops as repeated steps that have iteration variables that change each time through a loop. To understand this section it's important to focus on these topics in the lecture: While-loops (indefinite), For-loops (definite), Infinite loops, iteration variables, breaks, continues, and idioms. You'll be quizzed on the material and given an assignment on it as well. See appendix B.5 for more information.

Course 2: Python Data Structures

Course 2 is broken down into 7 weeks, 5 quizzes, and 7 assignments, just like course 1. Similar to the previous course, if you watched the recommended Python tutorial on Youtube then this course should also be a breeze as all the topics in the course were covered in the tutorial. Week 1, "Ch.6: Strings," introduces the concepts of strings and string manipulation with functions. It's easiest to understand strings as a sequence of characters (letters and integers) that are defined by quotes. To understand this chapter it's important to focus on these topics during the lecture videos: String type, Read/Converting, Indexing Strings [], Slicing Strings [2:4], looping through strings (for and while), concatenation (+), string operations, string library, comparisons,

search/replace text, and stripping whitespace, as you'll be quizzed and given an assignment. See appendix C.1 for more information. As for Week 2, "Unit: Installing and Using Python," feel free to skip this material as it's the exact same thing that was shown in week 2 of course 1.

Week 3, "Ch.7: Files," has to do with Python and external files and how to process them, such as opening them, reading them, or writing to them. To understand how to use python with files you must focus on these topics: Opening a file - file handle, file structure - newline character, reading files line by line (for loop), searching for lines, reading file names, and dealing with bad files. As always you'll be quizzed material, and this week will be given two assignments on the concepts. See appendix C.2 for more information.

Week 4, "Ch.8: Lists," goes into everything you need to know about lists, specifically how to manipulate them, and the relations between lists and strings. Lists are very similar to Arrays in Java. Lists are best thought of as a collection, they allow us to put many values into a single variable. Pay attention to the following topics during the lecture videos: Lists and definite loops (for), indexing and lookup, list mutability (elements can be changed with an index operator), function (len, min, max, sum), slicing and sorting lists, parsing, appending/removing, and splitting strings into lists. You will be quizzed on the concepts above (syntax and definition) as well given to assignments on them. See appendix C.3 for more information.

Week 5, "Ch.9: Dictionaries," it's no coincidence that the course introduces dictionaries next , given that they're very similar. Like lists, dictionaries are collections. However, whereas a list is a linear collection of values that stay in order, dictionaries are like a bag of values, each with its own label or key. Dictionaries are Python's most powerful data collection, if you understand them they can prove useful for fast database-like operations. This week covers dictionaries as a whole with emphasis on counting dictionaries, and the relation between dictionaries and files. Focus on these topics from the lecture videos as you'll be quizzed and given an assignment: Lists vs Dictionaries, Dictionary constants, the get()" method, the most common word, hashing & lack of order, writing dictionary loops, sorting dictionaries, and sneak peek: tuples. See appendix C.4 for more information.

Week 6, "Ch.10: Tuples," another concept similar to lists, tuples are used to store multiple values although tuples are immutable; meaning once you create one, you can't alter its contents (like a string). Tuples, unlike lists are also comparable (>, <, =). Tuples are useful when working with data that needs to be immutable and comparable, like coordinates for example. To understand tuples focus on these topics: tuple syntax, immutability, comparability, sorting, tuples in assignment statements, and sorting dictionaries by key or value. Be prepared for a quiz and an assignment on the topics above. See appendix C.5 for more information.

Week 7, “Graduation,” marks the end of the Python for Everybody Specialization’s “halfway mark.” Feel free to skip this section if necessary. There’s no quizzes or assignments, just a “Graduation” video, and a video that briefly explains what’s to come in the second half of the specialization.

Course 3: Using Python to Access Web Data

Course 3 is broken down into 6 weeks, 5 quizzes, and 8 assignments. Unlike the two previous courses, expect to start spending more time on the last three courses as little of the content was covered in the Python Youtube tutorial. Week 1, “Getting Started,” is the only section in this course that you should skip through. The content is the same as week 2 from the previous courses, the only video worth watching is the one on the new “Peer-graded” way to submit assignments, instead of the python playground. Week 2, “Ch.11: Regular Expressions,” thoroughly covers regular expressions, also referred to “Regex” or “Regexp,” they provide a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters.[1] To understand regular expressions it important to focus on these topics during lecture: Regular Expression quick guide, the regular expression module (import re, re.search(), re.findall()), wild-card characters, matching and extracting data, Double Split pattern vs Regex version, and the escape character. As always you’ll be quizzed and given an assignment on the topics above. See appendix D.1 for more information.

Week 3, “Ch.12: Networks and Sockets,” introduces web data and network architecture by focusing on networked technology, Hypertext Transfer Protocol (HTTP), and sockets (an endpoint of a bidirectional inter-process communication flow across and Internet protocol-based network). To understand this week’s material focus on these lecture topics: TCP and Application Protocols (HTTP), TCP connections/Sockets, TCP port numbers, Sockets in Python, and Getting Data From the Server. Expect to be quizzed on the material above, as well as an assignment that puts it to use. See appendix D.2 for more information.

Week 4, “Ch.12: Programs that Surf the Web,” goes further into web exploration and explains more efficient ways to receive web data. This week goes into Unicode Characters & Strings, Retrieving/Parsing web pages, and using **urllib** & Beautifulsoup. Pay attention to these topics during lecture: ASCII (characters and strings), HTTP request, **urllib** (Python library that does all the socket work for us, web pages look like files), following links, Parsing HTML (web scraping), and Beautifulsoup (Python package for parsing HTML and XML documents, and extracting data from HTML). After the

[1] http://en.wikipedia.org/wiki/Regular_expression

lecture you'll have a quiz and two assignments to complete regarding the material. See appendix D.3 for more information.

Week 5, "Ch.13: Web Services and XML," focuses on the first half of chapter 13 which mainly focuses on XML, one of the two commonly used formats for data going between applications across networks (JSON is the other). Extensible Markup Language's (XML) main purpose is to help information systems share structured data. The lecture videos focus on the type of data on the web and XML, specifically the XML schema and parsing XML. To understand this half of the chapter focus on these topics: XML elements (nodes) and basics, terminology (tags, attributes, serialize/deserialize), XML trees and paths, XML Schema (contract), XSD XML Schema (W3C spec), XSD structure, XSD data types, XSD constraints, and the ISO 8601 date/time format. You will have a quiz on those topics and an assignment as well. See appendix D.4 for more information.

Week 6, "Ch.13: JSON and REST Architecture," covers the second half of chapter 13 and focuses on JavaScript Object Notation (JSON), the Service Oriented Approach, Service Oriented Architectures, using Application Program Interfaces (APIs), and securing API requests. JSON's syntax is very similar to Dictionaries, to understand the structure I'd recommend reviewing Dictionaries. It will also help you to think of APIs as largely abstract in that it specifies an interface and controls the behavior of the objects specified in that interface.[2] Focus on the topics mentioned above during the lecture videos as you'll have a quiz to complete as well as two assignments. See appendix D.5 for more information.

Course 4: Using Databases with Python

Course 4 is broken down into 5 weeks, 5 quizzes, and 5 assignments. Week 1, "Object Oriented Python," goes over Object Oriented definitions/terminology, Classes and Objects, Object Life Cycle, and Object Inheritance (the ability to extend a class to make a new class). As a quick tip, it helps to think of a Class as a template, think of a Method/message as a defined capability of a class, think of a Field/attribute as a bit of data in a class, and think of an Object/instance as a particular instance of a class. You will be assigned a quiz so make sure to pay close attention to the topics mentioned above during lecture. There's no assignment this week so I referenced notes from the YouTube Python tutorial to help you understand the material, see appendix E.1 for more information.

Week 2, "Basic Structured Query Language," marks the introduction into another coding language, SQL. The introduction this week focuses on installing the SQLite

[2] http://en.wikipedia.org/wiki/API

Browser and the first concepts from chapter 15: Relational Databases, Using Databases, and Single Table CRUD (Create, Read, Update, Delete). To understand this week's material focus on these concepts from the lecture: Terminology (Database, Relation, Tuple, Attribute), CRUD, Database Administrator and Model, and SQL Syntax (Create, Insert, Delete, Update, Select, Order By). Prepare to be quizzed on the concepts and given two assignments. See appendix E.2 for more information.

Week 3, "Data Models and Relational SQL," we are diving further into chapter 15 where the focus is on designing data models, representing a data model in tables, inserting relational data, and reconstructing data with the JOIN operator. I recommend to focus on these topics during lecture: Database Design, Building a Data Model, Database Normalization (3NF), Integer Reference Pattern, Keys (Primary, Logical, Foreign), Relationship Building (in tables), and Using Join Across Tables. These concepts are important, they will teach you how to make programs and databases faster and efficient, as well as prepare you for the quiz and assignment. See appendix E.3 for more information.

Week 4, "Many-to-Many Relationships in SQL," covers the last portion of chapter 15. Many-to-Many relationships are what we need to model a connection where each side can have many values. There's not that much to them, focus on learning the syntax, and how to properly insert values into a many-to-many relationship, and you'll be set for the quiz and the assignment. See appendix E.4 for more information. It's important that you understand the material from chapter 15 because if you do, it will allow you to scale to large amounts of data.

Week 5, "Databases and Visualization," covers chapter 16 and focuses on GeoCoding, GeoCoding Visualization, and web crawling. This is the last material to learn in the course and the bulk of it is conceptual. There's not that much to learn as the material is mostly just combining everything you have learned (APIs, SQL, reading files, pulling web data, **urllib** to read HTML files, try and except, JSON, loops, dictionaries, parsing, if-statements, stripping/slicing, functions, and objects/inheritance). You are provided with all of the code for the lecture and the assignment, the lecture videos are really just code walkthroughs done by the instructor, Dr. Chuck, so all you have to do is pay attention and follow along. As for the assignment, all you do is one small edit then run the code, and submit screenshots of your results. See appendix E.5 for more information.

Course 5: Capstone: Retrieving, Processing and Visualizing data with Python

Course 5 is broken down into 7 weeks, 2 assignments (projects), and only 1 quiz. This course is very similar to week 5, "Databases and Visualization," from the last course. There's no new data to learn, you are covering the same concepts on

Visualization from Chapter 16. The course walks you through the two assignments, which are similar to the one in week 5 from course 4. You'll be provided with all of the code, and the lecture videos are Dr. Chuck walking you through all of the code for the assignments, all you have to do is pay attention, run the code, and submit the requested screenshots. This course is all about combining all of the concepts you have learned throughout the course (APIs, SQL, reading files, pulling web data, **urllib** to read HTML files, try and except, JSON, loops, dictionaries, lists, parsing, web-scraping & parsing with BeautifulSoup, if-statements, stripping/slicing, functions, Regular expressions, encoding/decoding, and objects/inheritance), and since Dr. Chuck explains the assignments in the lecture, there's really no more advice I can give that I haven't already.

Week 1, "Welcome to the Capstone," introduces you to the last course and briefly explains the two assignments that will be lectured on. One of the differences between the last visualization assignment we did and these projects is that instead of reading data from files we will be reading data directly from web pages (url's). So, since these projects will involve more web-scraping, this week you are having a review quiz (already taken in course 4) to refresh you on internet topics like: UTF-8, ASCII, unicode, and encoding/decoding data. Aside from the review material, feel to skip through the rest of week 1.

Week 2, "Building a Search Engine," introduces us to the first project, a search engine. All the code required for the search engine has already been completed and is included in the "pagerank" folder within the "Sample code" folder that was downloaded at the beginning of the specialization. The"pagerank" folder should contain the following files: spider.py, sprank.py, spreset.py, spjson.py, spdump.py, spider.js, force.js, force.html, and d3.v2.js. The lecture videos break up the Pagerank assignment code (search engine) into three parts; Spidering, Computation, and Visualization. The Spidering part uses the spider.py code to create the spider.sqlite database, connects to a url web page (user-decides), then "spiders" or web crawls for a number (user decides) of subpages, and inserts the data into the database. The Computation part uses the sprank.py code, which contains the "Page Rank" algorithm, the code reads the spider.sqlite database, retrieves web page ID's, and ranks them in terms of visitation frequency, essentially looking for the most popular/useful pages. The database records two ranks, (New & Old Rank), because let's say we chose to "spider" 10 pages and then later wanted to "spider" 30 pages, the previous ranks will need to be recalculated with the algorithm to account for the new pages. The Visualization part uses the spdump.py code to read the spider.sqlite database and uses a Select statement to retrieve the web page counts (frequencies), old rank, new rank, web page ID, and the url for all of the web pages, then prints it to the screen. The Visualization part also uses the spjson.py code to look at all that data and produce a JavaScript file. It's going to write a JavaScript

file, that is fed into the visualization, using d3 (Javascript library used for visualization, this is provided as this is not covered in the Python for Everybody Specialization). Spjson.py joins the links with the webpages, where HTML is not null, or error is not null, in order by the number of inbound links. So, we're looking at the webpages (user decides how many nodes) that have the highest number of inbound links. Spjson.py creates the spider.js file which gives all the information on the website nodes as well as the dimensions (circle weight and line thickness) which will be fed to the force.html file that will link us to the visualization. The force.html file also uses the force.js file (which contains the d3 javascript code that draws the circles and defines the circle colors, and makes the circles bigger and smaller, and connects all the lines in between the circles) as it's visualization source. So spider.js is also feeding the force.js file. So when we're all done running the code, we simply open force.html and select a browser to view the visualization of the data. See appendix F.1 for screenshots of the project.

Week 3, "Exploring Data Sources (Project)", provides a break in between assignments. This week you are simply provided with a list of different common data sources that you could use on the Page Rank code. There's also an optional choice to post to a discussion post on one of the provided databases. Feel free to skip through this week if desired.

Week 4, "Spidering and Modeling Email Data," introduces us to the first half of the second project, the second half of the project is done during week 6. The second project is very similar to the previous one, except instead of spidering web pages, this time we will spider email data, about 1gb of it. Just like the Page Rank project/assignment, all the code required for this project has already been completed and is included in the "gmane" folder within the "Sample code" folder that was downloaded at the beginning of the specialization. The "gmane" folder should contain the following files: gmane.py, gmodel.py, mapping.sqlite, gbasic.py, gline.py, gline.js, gline.htm, gword.py, gword.js, gword.htm, d3.v2.js, and d3.layout.cloud.js. The lectures for the whole Gmane assignment code are also broken into three parts: Retrieval (similar to Spidering), Modeling (similar to Computation), and Visualization. This week we only focus on Retrieval and Modeling. The Retrieval part uses the gmane.py file to create the content.sqlite database, connects to mbox.dr-chuck.net, then "spiders" the email data (gmane.py uses re.findall and Regex to parse the data they need), then inserts the data into the content.sqlite database. The Modeling part uses the gmodel.py file to clean up the data (as it's still fairly raw email data) that was retrieved by using the gmane.py code. The gmodel.py code creates a new database called index.sqlite, connects to the mapping.sqlite database (created by Dr. Chuck to track people who changed email addresses, maps the old emails to the new ones, don't need to worry about this so much) to create a dictionary of the data. The gmodel.py code then reads the content.sqlite database and continues to use Regex and stripping to parse the data

as well as using zlib to compress some of the data, then inserts the new data into the index.sqlite database. This concludes the week 4 material, see appendix F.2 for screenshots of the code.

Week 5, “Accessing New Data Sources (Project),” provides you with another break. This week there’s another optional discussion board on using data sources, as well as one of Dr.Chuck’s non-Python related videos. Feel free to skip through this week if desired.

Week 6, “Visualizing Email Data,” we are picking up right back where we left off from week 4, visualization. The Visualization part uses the gbasic.py code to read the index.sqlite database, creates dictionaries (senders, messages, sendcounts, sendorgs) as it’s faster than using a Select statement in SQL, then uses a for-loop to go through all the messages, sorts them (sorted function), and then prints out the top email list participants and organizations (amount depends user). The Visualization part also uses the gword.py code to read the index.sqlite database and use the data to produce the gword.js (Java Script) file which will contain all the dimension and design information for the d3 visualization files, which will then be fed to the gword.htm file. Once we’re done running all the code, we just open the gword.htm file and select a browser to be linked to the “Word Cloud” visualization. Similarly, we will follow the same steps, but with different code to create a different visualization. This time we will use the gline.py (breaks the data into months for the top 10 organizations) code to produce the gline.js file which contains the dimension and design information for the d3 visualization files, which will then be fed to the gline.htm file. Then after running the code we can open the gline.htm file and select a browser to be linked to the “Time-Line” visualization which represents developer participation by month. See appendix F.3 for screenshots of the code.

Week 7, “Visualizing New Data Sources (Project)”, congratulations, you made it to the end of the course, you can now relax. This week’s material is just another optional discussion board asking you to reflect on the assignment and what experience you think was gained by taking the Python for Everybody Specialization.

Upon the successful completion of each course in the specialization you will receive your certificate. Once obtaining all five certificates you will have earned the Python for Everybody Specialization certification. See appendix to view my certifications, hopefully seeing mine will encourage you to get your own. Python is a very sought out skill in the tech industry, those who have a thorough understanding of the language are only benefiting themselves, and setting themselves up for success.

Appendix

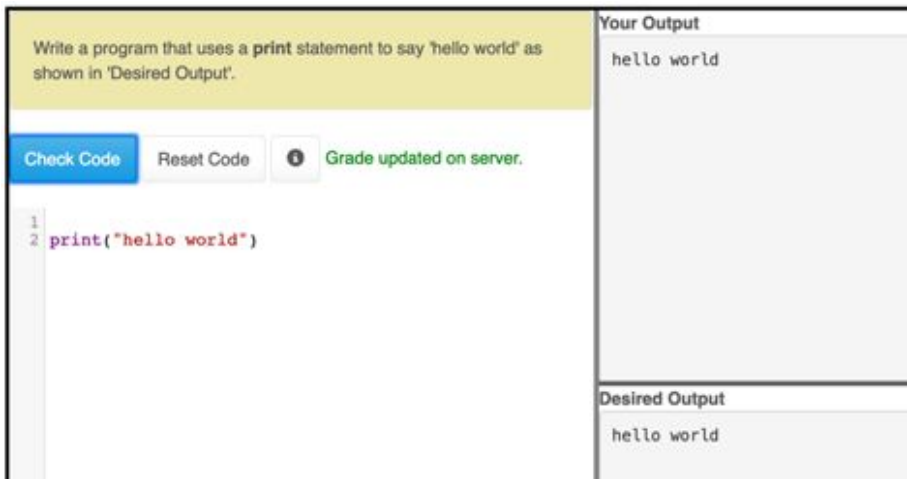
A. Work Log:

Date	Start Time	End Time	Hrs	Activity
1/8/19	8pm	12am	4	Research: creating WinPE USB, how to load it on a server
1/10/19	6pm	7pm	1	Met with Best Buy, ultimately rejected initial project idea
1/10/19	5pm	6:30pm	1.5	Met with OfficeMax, ultimately rejected initial project idea
1/17/19	8am	8:30am	0.5	Research: Found out about the Coursera Python Specialization
1/18/19	8pm	9pm	1	Submitted financial aid application for Coursera
2/7/19	3pm	9pm	6	Watched Python Youtube tutorial, had to pause/replay for notes
2/11/19	10am	11am	1	Coursera – Chapter 1: Why We Program
2/20/19	4pm	6:30pm	2.5	Reviewed Youtube tutorial, quick review to refresh
2/21/19	7pm	12am	5	Course 1: Week 1, Week 2, Week 3, Week 4
2/23/19	6pm	11am	5	Course 1: Week 5, Week 6, Week 7
3/14/19	4pm	6:10pm	2.17	Course 2: Week 1
3/18/19	5pm	9pm	4	Course 2: Week 2, Week 3. Week 4
3/23/19	8:30pm	12:30am	4	Course 2: Week 5, Week 6, Week 7
4/26/19	5pm	10pm	5	Course 3: Week 1, Week 2
4/27/19	8am	12:30pm	4.5	Course 3: Week 3
5/5/19	12pm	6pm	5	Course 3: Week 4
5/19/19	5pm	8pm	3	Course 3: Week 5
5/20/19	8am	10am	2	Course 3: Week 5
8/5/19	12pm	6:20pm	4	Took a break, needed to review Youtube tutorial and Course 3
8/6/19	8am	1pm	5	Course 3: Week 6
8/8/19	10am	5pm	5	Course 4: Week 1, Week 2
8/10/19	12pm	5pm	4	Course 4: Week 3
8/13/19	6pm	10pm	4	Course 4: Week 4
8/17/19	8pm	11:30	3.5	Course 4: Week 5

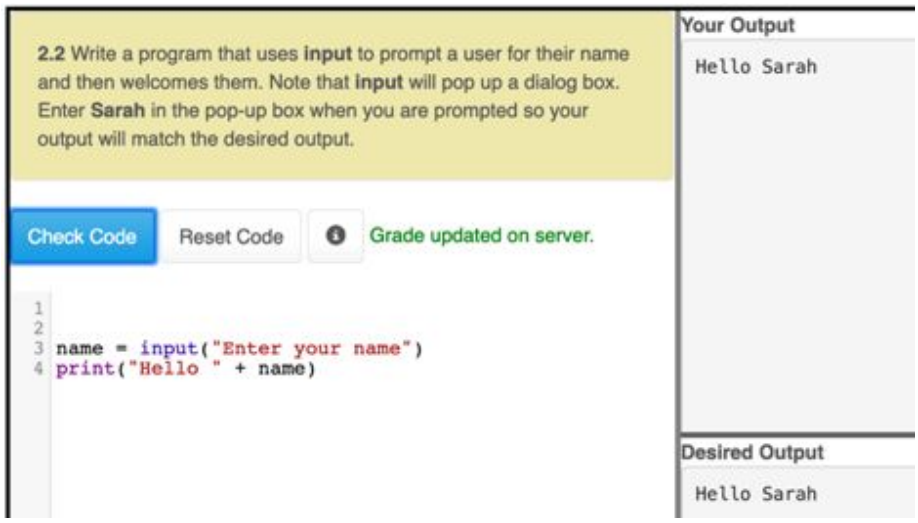
8/19/19	8am	12pm	3	Course 5: Week 1, Week 2, Week 3
8/23/19	8pm	1:30pm	4	Course 5: Week 4, Week 5, Week 6, Week 7
8/28/19	9pm	11am	2	Started Final Report
8/29/19	7pm	11am	2	Final Report: Had to re-review early Python
8/30	10pm	12am	2	Final Report: Had to re-review early Python
8/31	9am	11pm	2	Final Report: Had to re-review Python
9/1	10am	2pm	4	Final Report
9/2	4pm	8pm	4	Final Report
9/3	12pm	2pm	2	Final Report
9/3	11pm	10:30am	11.5	Finishing Final Report
Total Amount of Hours = 121.17				

B. Course 1:

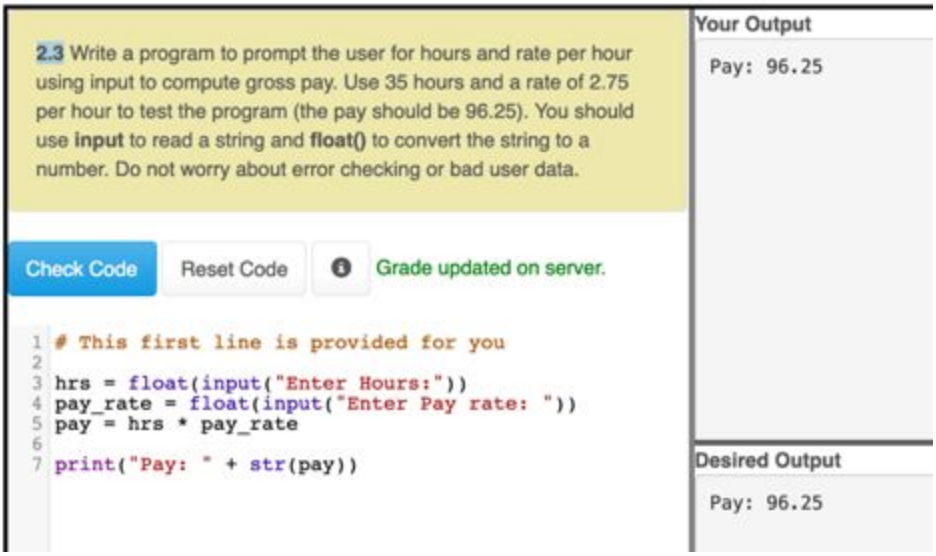
B.1: Week 3 - Hello World Assignment: How to print text out to the screen, for integers, no quotes required.



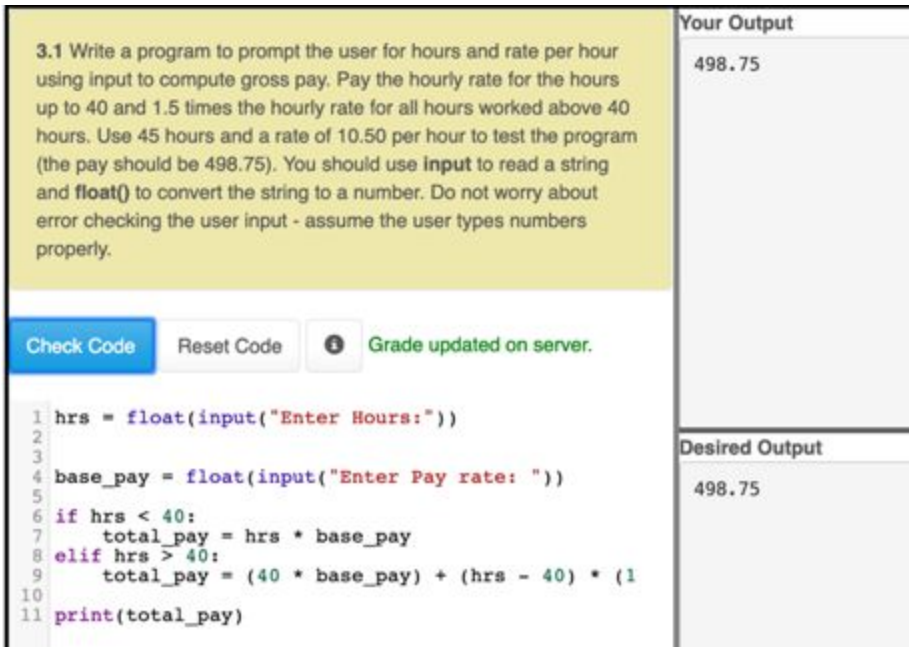
B.2: Week 4 - Assignment 2.2: Variables (name), how to get user input (input), and combining a string and variable with an expression.



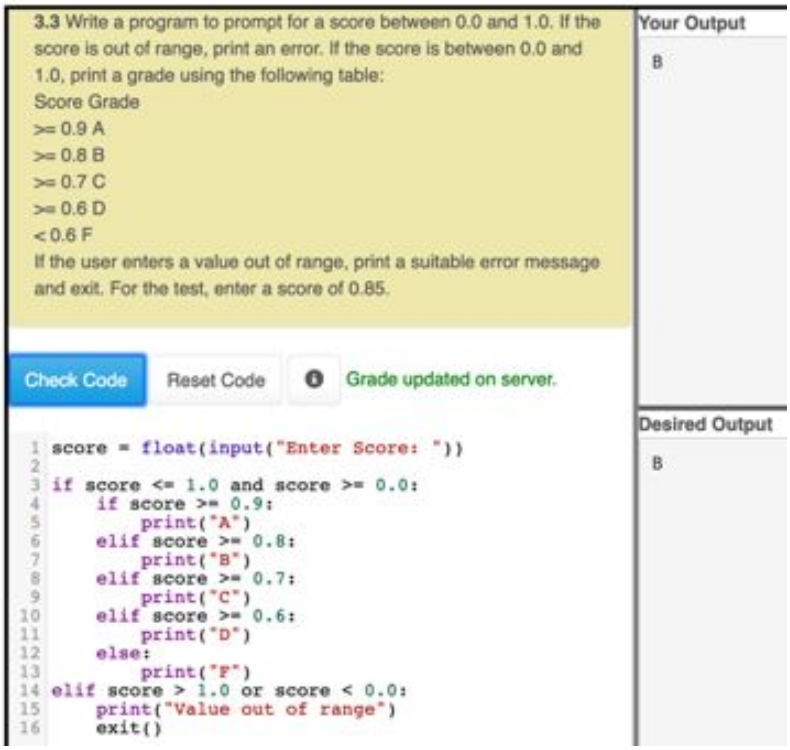
Week 4 - Assignment 2.3: In addition to the use of variables and expressions, requires conversion between types, thus the use of float(input()) to change a string into an integer, and str(pay) to do the opposite. Also introduces operators for multiplication to calculate pay.



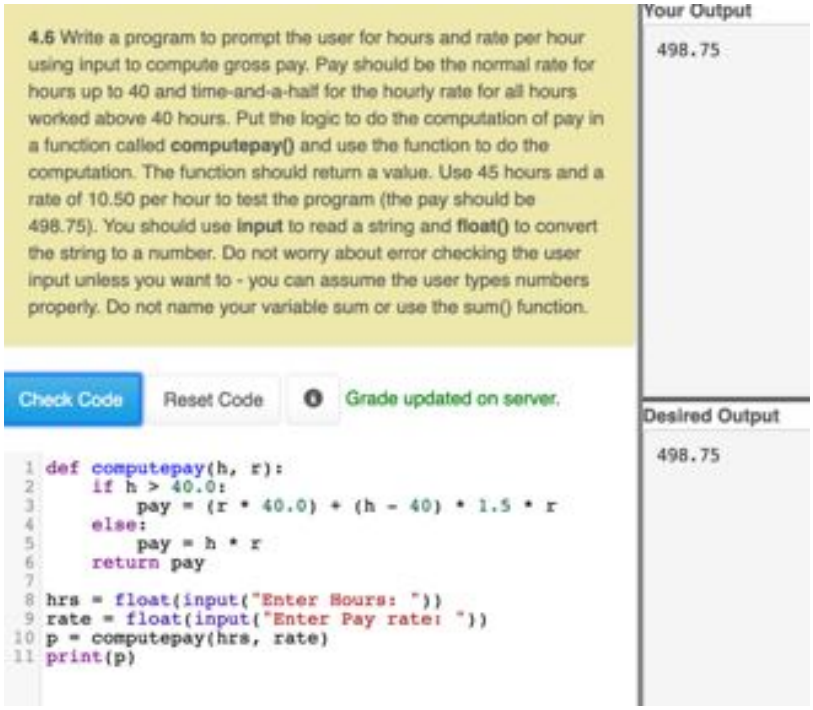
B.3: Week 5 - Assignment 3.1: Introduces basic If statements (If- elif) to improve the code above by taking into account the condition of overtime to get an accurate calculation. Two-way decision, also includes the use of comparison operators (<,>).



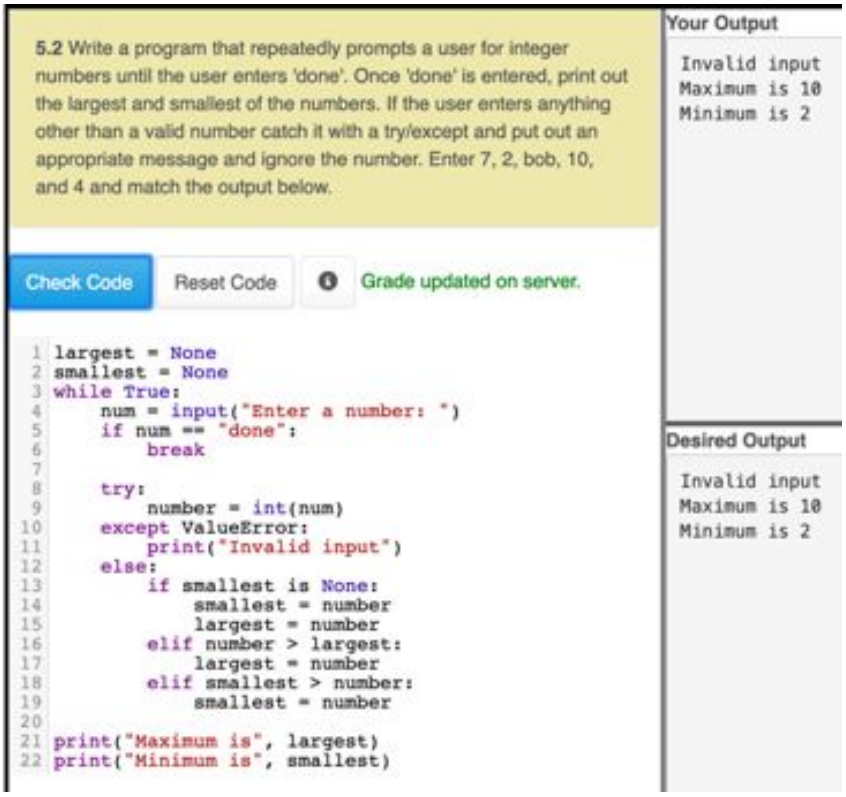
Week 5 - Assignment 3.3: Multi-way decisions and nested If-statements used to calculate grade.



B.4: Week 6 - Assignment 4.6: Functions, how to define them (def), set parameters (h, r/ hrs, rate) and how to call it (computepay(hrs,rate)). Now instead of repeating code to compute the pay, can just call the function.

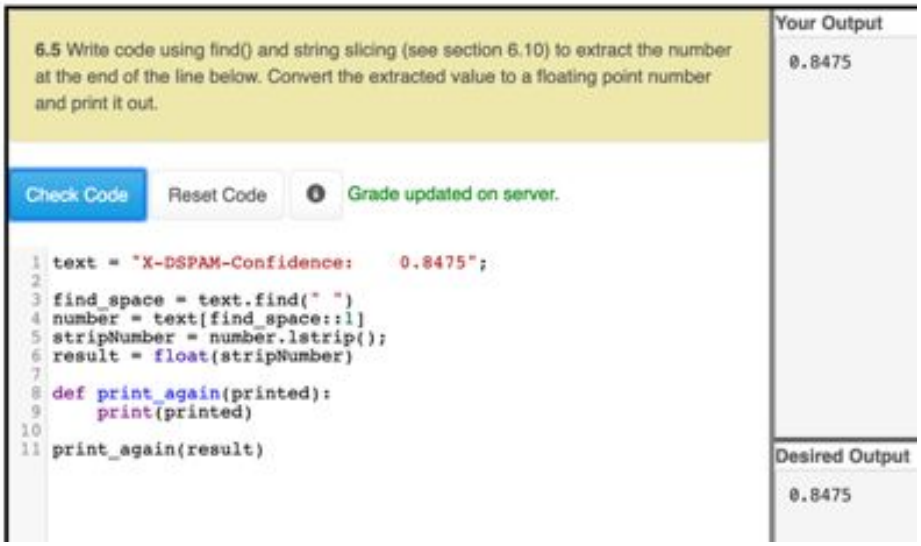


B.5: Week 7 - Assignment 5.2: Use of loops (while-loop), breaks, try/except, and a boolean to create the described program.

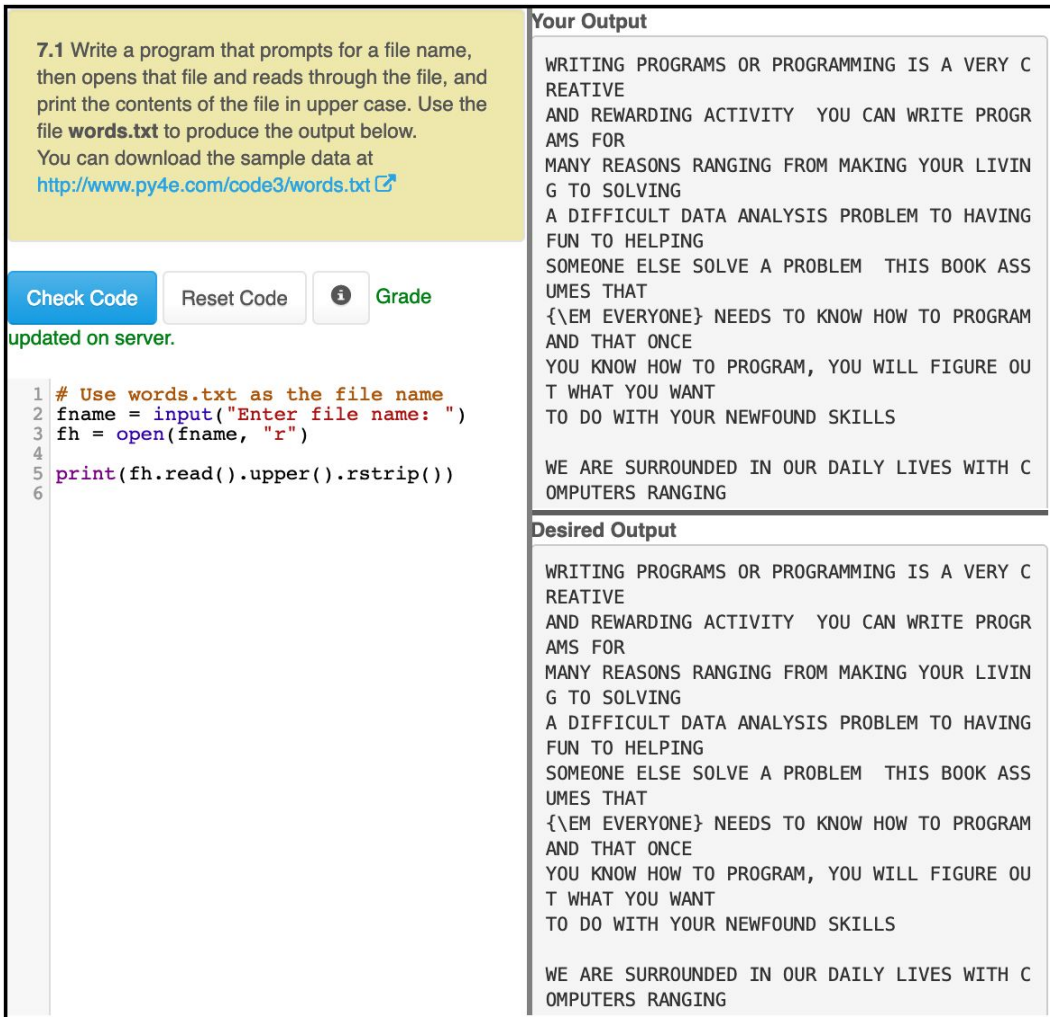


C. Course: 2

C.1: Week 1 - Assignment 6.2: Focusing on strings, the assignment has us use the find() function to any white spaces in the text, indexing the white space, and then stripping away the whitespace from the left side by using lstrip().



C.2: Week 3 - Assignment 7.1: Writing a simple program to open and read a file (open(fname, “r”)), and then print the file line by line (.read() function), in all caps (.upper() function), and rid of any white space after the text (.rstrip() function. Sample data is provided in the sample code download.



7.1 Write a program that prompts for a file name, then opens that file and reads through the file, and print the contents of the file in upper case. Use the file **words.txt** to produce the output below.
You can download the sample data at
http://www.py4e.com/code3/words.txt

Check Code | Reset Code | Grade

updated on server.

```
# Use words.txt as the file name
fname = input("Enter file name: ")
fh = open(fname, "r")

print(fh.read().upper().rstrip())

```

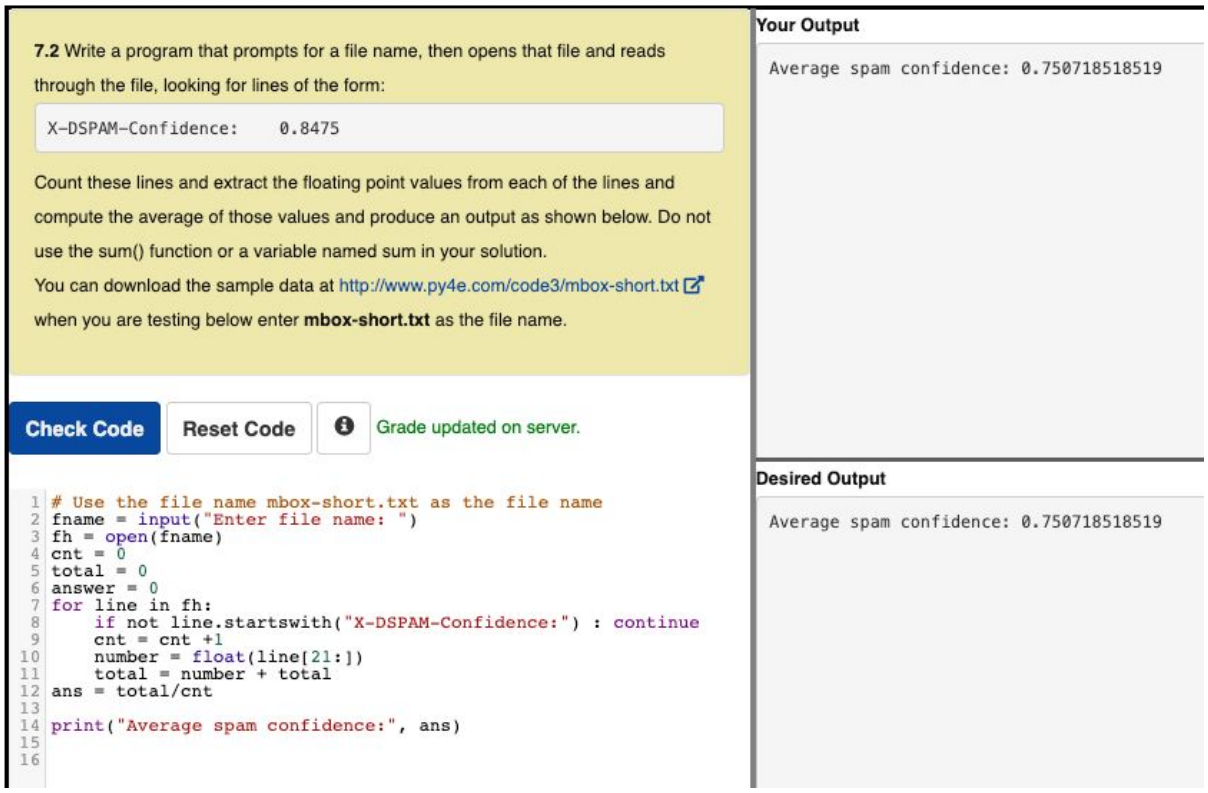
Your Output

```
WRITING PROGRAMS OR PROGRAMMING IS A VERY C
REATIVE
AND REWARDING ACTIVITY  YOU CAN WRITE PROGR
AMS FOR
MANY REASONS RANGING FROM MAKING YOUR LIVIN
G TO SOLVING
A DIFFICULT DATA ANALYSIS PROBLEM TO HAVING
FUN TO HELPING
SOMEONE ELSE SOLVE A PROBLEM  THIS BOOK ASS
UMES THAT
{\EM EVERYONE} NEEDS TO KNOW HOW TO PROGRAM
AND THAT ONCE
YOU KNOW HOW TO PROGRAM, YOU WILL FIGURE OU
T WHAT YOU WANT
TO DO WITH YOUR NEWFOUND SKILLS

WE ARE SURROUNDED IN OUR DAILY LIVES WITH C
OMPUTERS RANGING
```

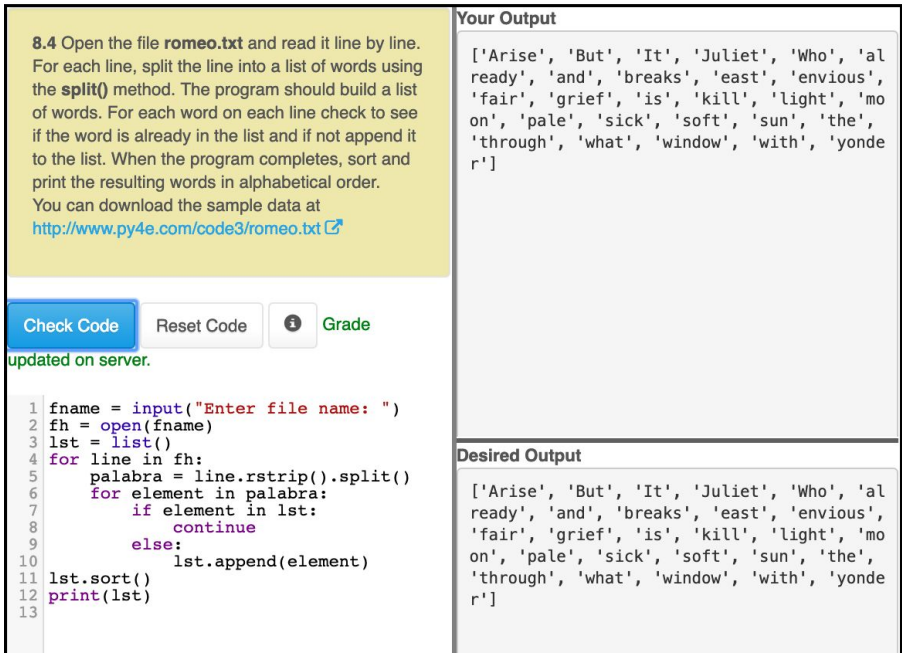
Desired Output

```
WRITING PROGRAMS OR PROGRAMMING IS A VERY C
REATIVE
AND REWARDING ACTIVITY  YOU CAN WRITE PROGR
AMS FOR
MANY REASONS RANGING FROM MAKING YOUR LIVIN
G TO SOLVING
A DIFFICULT DATA ANALYSIS PROBLEM TO HAVING
FUN TO HELPING
SOMEONE ELSE SOLVE A PROBLEM  THIS BOOK ASS
UMES THAT
{\EM EVERYONE} NEEDS TO KNOW HOW TO PROGRAM
AND THAT ONCE
YOU KNOW HOW TO PROGRAM, YOU WILL FIGURE OU
T WHAT YOU WANT
TO DO WITH YOUR NEWFOUND SKILLS

WE ARE SURROUNDED IN OUR DAILY LIVES WITH C
OMPUTERS RANGING
```

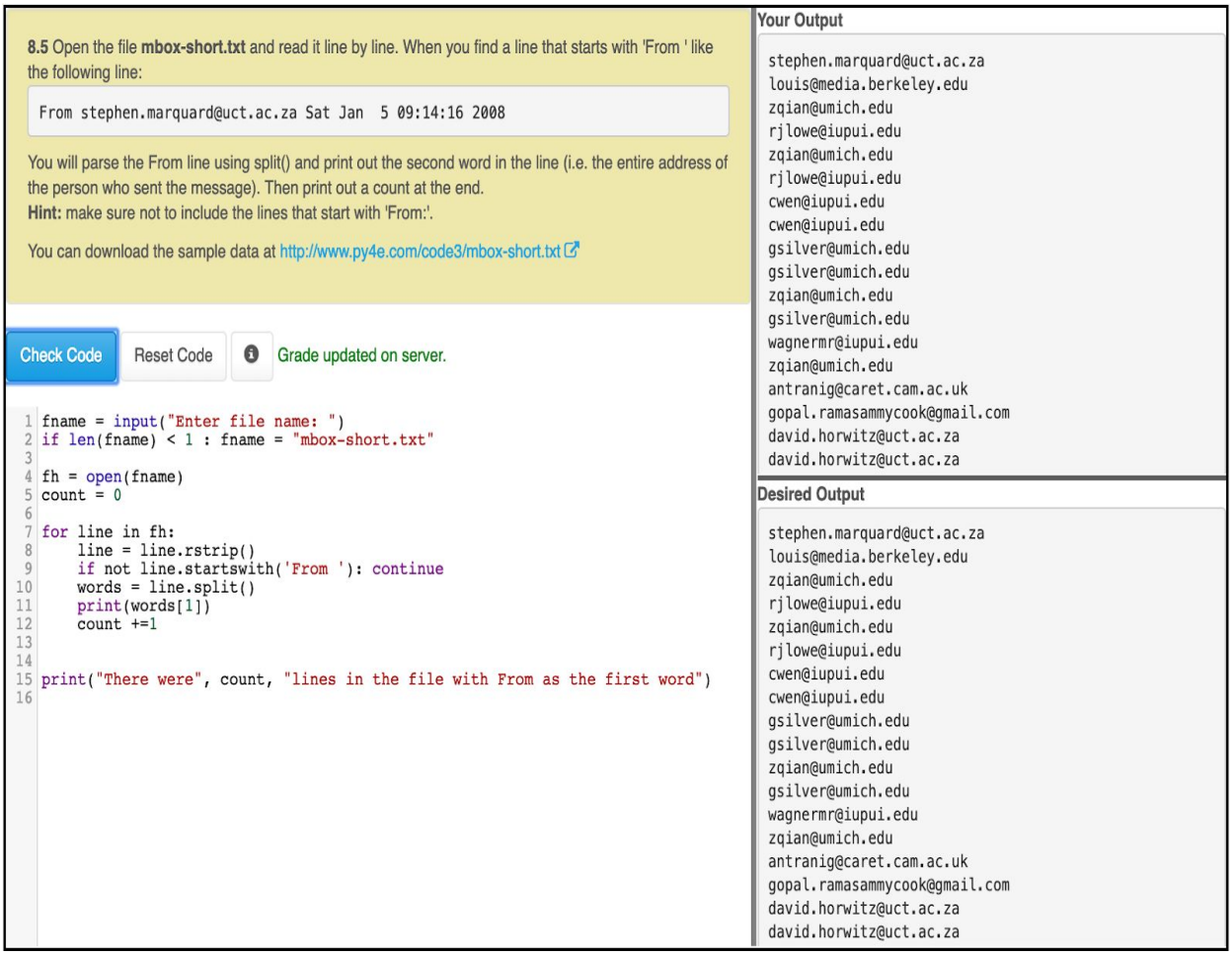
Week 3 - Assignment 7.2: Requires the use of a for loop to go through each line in the file looking for the specified text, while counting the number of lines (cnt) and calculating the average of those values (ans = total / cnt). Sample data is provided in the sample code download.



C.3: Week 4 - Assignment 8.4: Creating a list (lst), using a for loop to go through the file and separate strings into a list of words (palabra = line.rstrip().split()), adding those words (element) into the list we created (lst.append(element)) and alphabetizing it (lst.sort()).



Week 4 - Assignment 8.5: Using a for loop with an if statement to read through the file looking for lines (strings) that start with “From:” (if not line.startswith(“From: ”): continue), creating a list of them (words = line.split()), and printing out the strings with the “From:” removed by slicing them (print(words[1]).



8.5 Open the file **mbox-short.txt** and read it line by line. When you find a line that starts with 'From ' like the following line:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

You will parse the From line using split() and print out the second word in the line (i.e. the entire address of the person who sent the message). Then print out a count at the end.
Hint: make sure not to include the lines that start with 'From:'.

You can download the sample data at http://www.py4e.com/code3/mbox-short.txt

Check Code | Reset Code | Grade updated on server.

```
fname = input("Enter file name: ")
if len(fname) < 1 : fname = "mbox-short.txt"

fh = open(fname)
count = 0

for line in fh:
    line = line.rstrip()
    if not line.startswith('From '): continue
    words = line.split()
    print(words[1])
    count +=1


print("There were", count, "lines in the file with From as the first word")

```

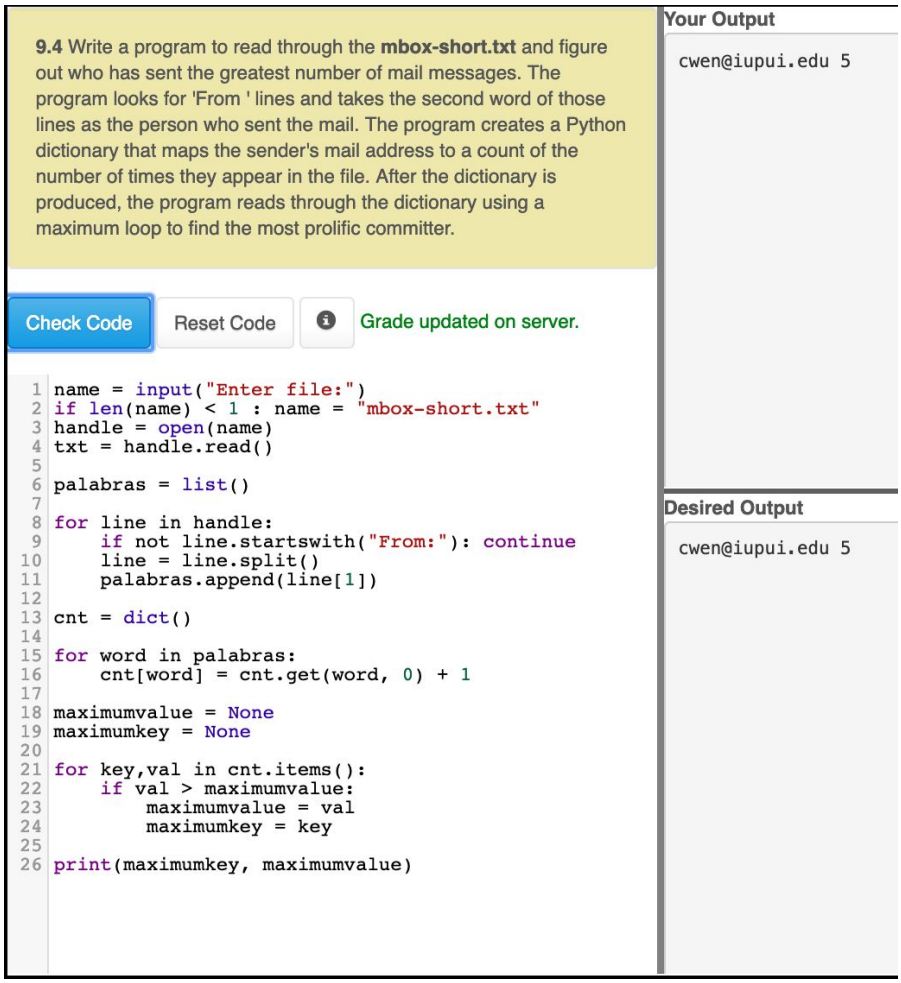
Your Output

```
stephen.marquard@uct.ac.za
louis@media.berkeley.edu
zqian@umich.edu
rjlowe@iupui.edu
zqian@umich.edu
rjlowe@iupui.edu
cwen@iupui.edu
cwen@iupui.edu
gsilver@umich.edu
gsilver@umich.edu
zqian@umich.edu
gsilver@umich.edu
wagnermr@iupui.edu
zqian@umich.edu
antranig@caret.cam.ac.uk
gopal.ramasammycook@gmail.com
david.horwitz@uct.ac.za
david.horwitz@uct.ac.za
```

Desired Output

```
stephen.marquard@uct.ac.za
louis@media.berkeley.edu
zqian@umich.edu
rjlowe@iupui.edu
zqian@umich.edu
rjlowe@iupui.edu
cwen@iupui.edu
cwen@iupui.edu
gsilver@umich.edu
gsilver@umich.edu
zqian@umich.edu
gsilver@umich.edu
wagnermr@iupui.edu
zqian@umich.edu
antranig@caret.cam.ac.uk
gopal.ramasammycook@gmail.com
david.horwitz@uct.ac.za
david.horwitz@uct.ac.za
```

C.4: Week 5 - Assignment 9.4: Similar to the previous code, after creating the list (palabras) of the second word of those lines, we create a dictionary (cnt = dict()) that maps the sender's email to a count of the number of times they appear (2nd for loop). The last for loop reads through the dictionary to find the most common sender.



9.4 Write a program to read through the **mbox-short.txt** and figure out who has sent the greatest number of mail messages. The program looks for 'From ' lines and takes the second word of those lines as the person who sent the mail. The program creates a Python dictionary that maps the sender's mail address to a count of the number of times they appear in the file. After the dictionary is produced, the program reads through the dictionary using a maximum loop to find the most prolific committer.

Check Code | Reset Code | Grade updated on server.

```
name = input("Enter file:")
if len(name) < 1 : name = "mbox-short.txt"
handle = open(name)
txt = handle.read()

palabras = list()

for line in handle:
    if not line.startswith("From:"): continue
    line = line.split()
    palabras.append(line[1])

cnt = dict()

for word in palabras:
    cnt[word] = cnt.get(word, 0) + 1

maximumvalue = None
maximumkey = None

for key,val in cnt.items():
    if val > maximumvalue:
        maximumvalue = val
        maximumkey = key

print(maximumkey, maximumvalue)
```

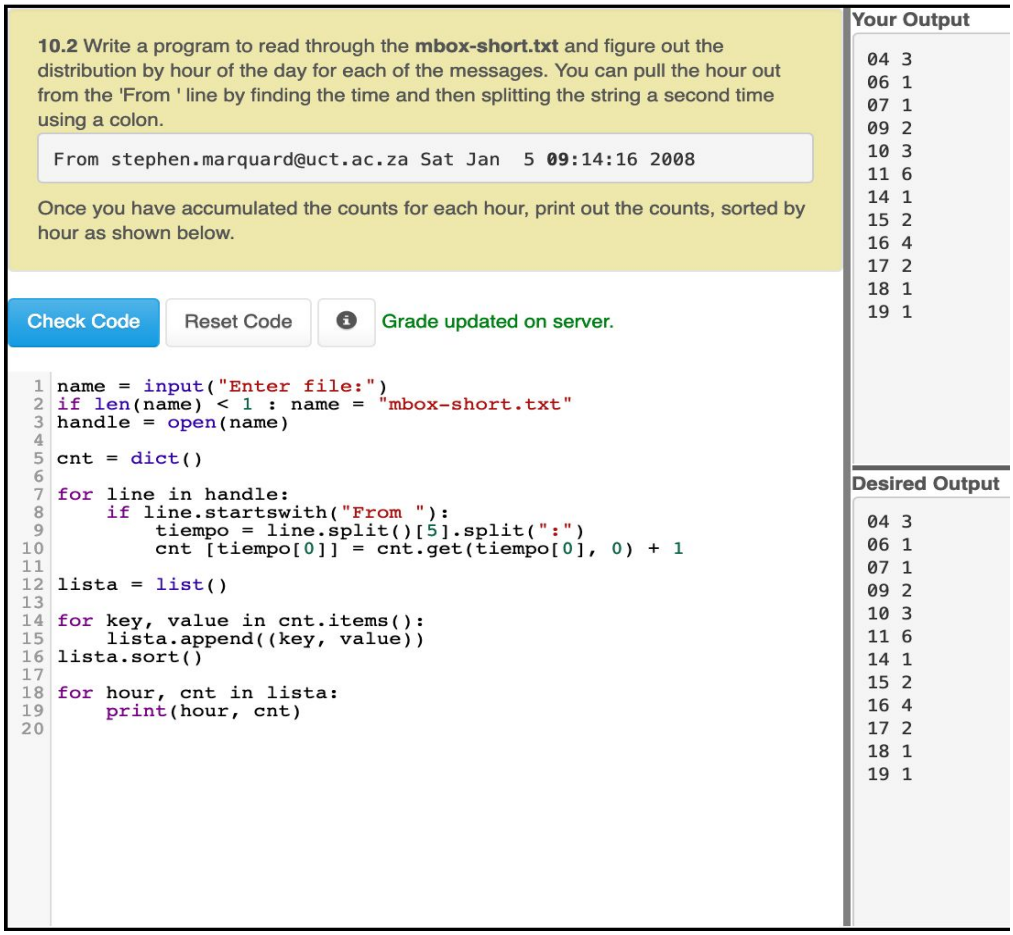
Your Output

```
cwen@iupui.edu 5
```

Desired Output

```
cwen@iupui.edu 5
```

C.5: Week 6 - Assignment 10.2: Tuples come into play after the first for loop,when the list (lista) is made. Tuples were utilized to input the key and value into the new list (lista.append((key, value))) that was created.



10.2 Write a program to read through the **mbox-short.txt** and figure out the distribution by hour of the day for each of the messages. You can pull the hour out from the 'From ' line by finding the time and then splitting the string a second time using a colon.

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

Once you have accumulated the counts for each hour, print out the counts, sorted by hour as shown below.

Check Code | Reset Code | Grade updated on server.

```
name = input("Enter file:")
if len(name) < 1 : name = "mbox-short.txt"
handle = open(name)

cnt = dict()

for line in handle:
    if line.startswith("From "):
        tiempo = line.split()[5].split(":")
        cnt [tiempo[0]] = cnt.get(tiempo[0], 0) + 1

lista = list()

for key, value in cnt.items():
    lista.append((key, value))
lista.sort()

for hour, cnt in lista:
    print(hour, cnt)
```

Your Output

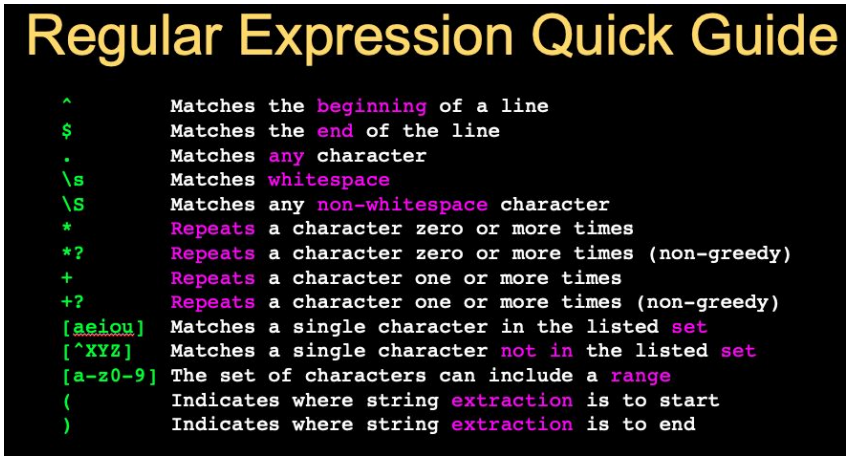
```
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
```

Desired Output

```
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
```

D. Course 3:

D.1: Week 2 - Extracting Data with Regular Expressions Assignment: Screenshot of the Regex quick guide from the lecture slides, may be difficult to memorize, so it's always a good idea to keep one handy. As the prompt says, you need to save the data files in the same folder where you code is saved or it will not work. After opening the file (hdl = open) and creating a list (numbers), Regex is used in the first for-loop, using re.findall() to extract portions of a string (in our case, integers) that match the regular expression, [0-9]+, which means one or more digits. Then the second loop sums it up.



Prompt for the assignment:

Welcome Alex Huerta from Using Python to Access Web Data

Your answer is correct, score saved. ×

Finding Numbers in a Haystack

In this assignment you will read through and parse a file with text and numbers. You will extract all the numbers in the file and compute the sum of the numbers.

Data Files
We provide two files for this assignment. One is a sample file where we give you the sum for your testing and the other is the actual data you need to process for the assignment.

- Sample data: http://py4e-data.dr-chuck.net/regex_sum_42.txt (There are 90 values with a sum=445833)
- Actual data: http://py4e-data.dr-chuck.net/regex_sum_189368.txt (There are 99 values and the sum ends with 821)

These links open in a new window. Make sure to save the file into the same folder as you will be writing your Python program. **Note:** Each student will have a distinct data file for the assignment - so only use your own data file for analysis.

Data Format
The file contains much of the text from the introduction of the textbook except that random numbers are inserted throughout the text. Here is a sample of the output you might see:

```
Why should you learn to write programs? 7746
12 1929 8827
Writing programs (or programming) is a very creative
7 and rewarding activity.  You can write programs for
many reasons, ranging from making your living to solving
8837 a difficult data analysis problem to having fun to helping 128
someone else solve a problem.  This book assumes that
everyone needs to know how to program ...
```

The sum for the sample text above is **27486**. The numbers can appear anywhere in the line. There can be any number of numbers in each line (including none).

Handling The Data
The basic outline of this problem is to read the file, look for integers using the **re.findall()**, looking for a regular expression of **'[0-9]+'** and then converting the extracted strings to integers and summing up the integers.

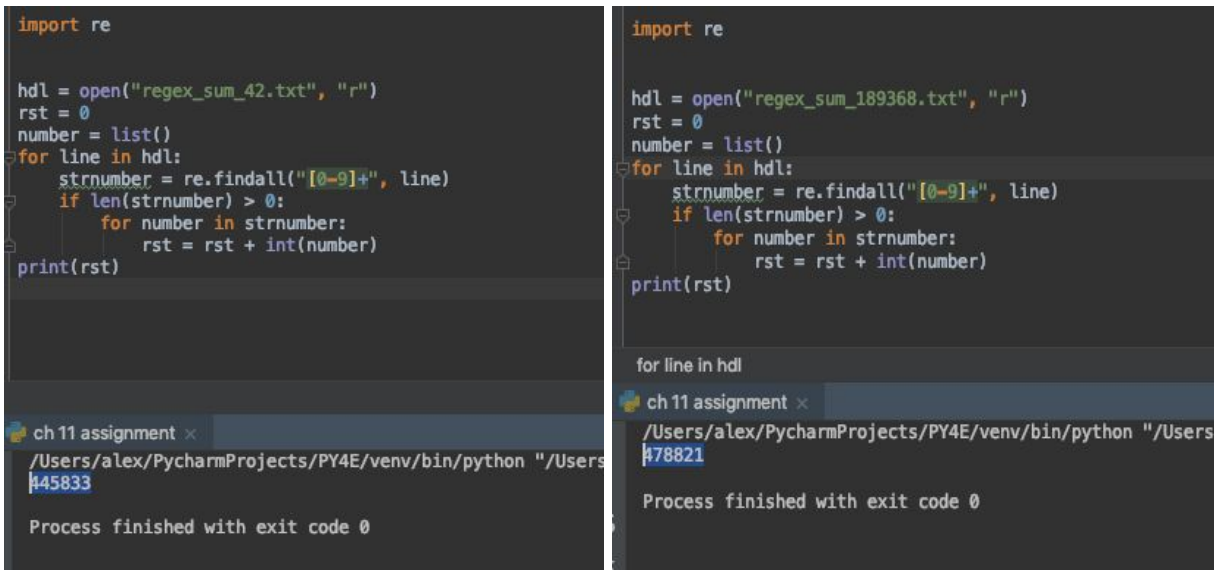
Turn in Assignent

Enter the sum from the actual data and your Python code below:
Sum: 478821 (ends with 821) Submit Assignment
Python code:

```
import re


hdl = open("regex_sum_189368.txt", "r")
rst = 0
number = list()
for line in hdl:
    strnumber = re.findall("[0-9]+", line)
    if len(strnumber) > 0:
        for number in strnumber:
            rst = rst + int(number)
print(rst)
```

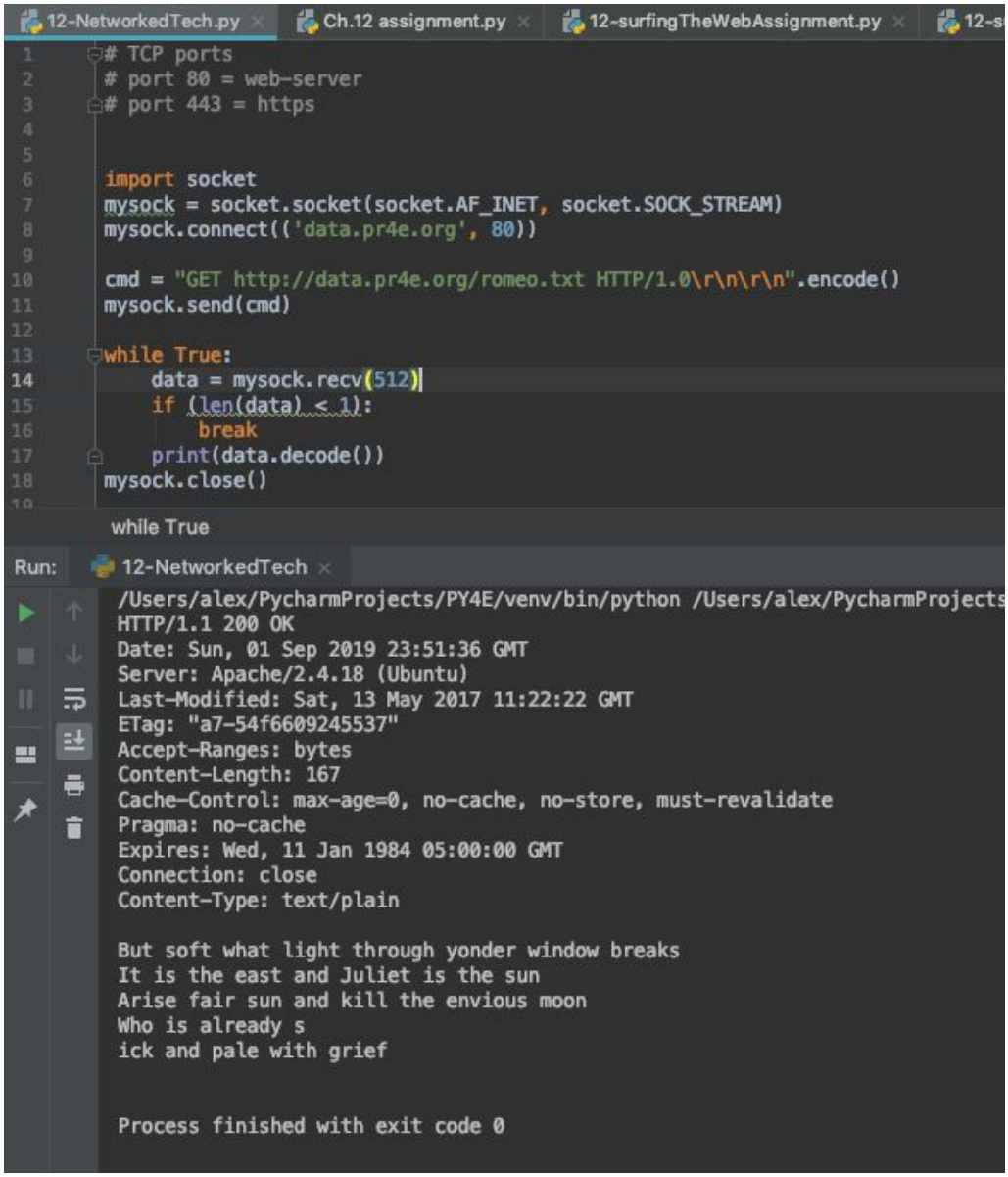
Select Language

IDE running my code on the sample data file (left) and on the actual data file (right) with the terminal displaying the output (highlighted):

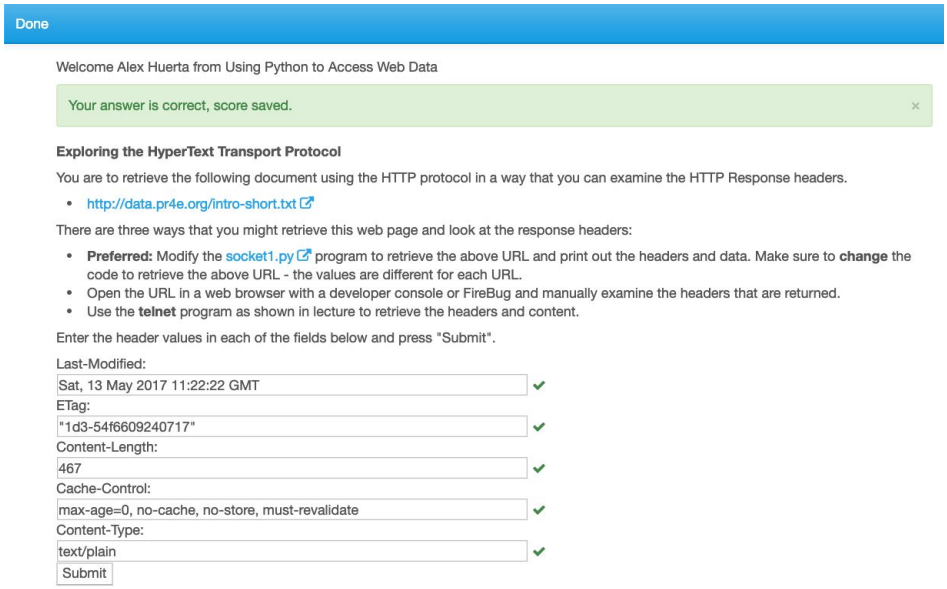


D.2: Week 3 - Understanding the Request/Response Cycle Assignment: First screenshot shows sample code that is provided to us during lecture. Sockets must be imported, and then defined (mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM), then you establish the connection to the host (data.pr4e.org) and port (80, web-server) you desire (mysock.connect((‘data.pr4e.org’, 80))). Then you create a variable (cmd) that stores a GET command for the exact webpage you want to get data on, as well as encoding it (.encode() in UTF-8 before sending the command to the host (mysock.send(cmd)). Use of a while-loop to retrieve the data (mysock.recv()), now the data retrieved will be in UTF-8, so it will need to be decoded to understand it (print(data.decode())). To retrieve the information needed for this week's assignment all that’s needed is to change the url address in the cmd variable to “http://data.pr4e.org/intro-short.txt”.

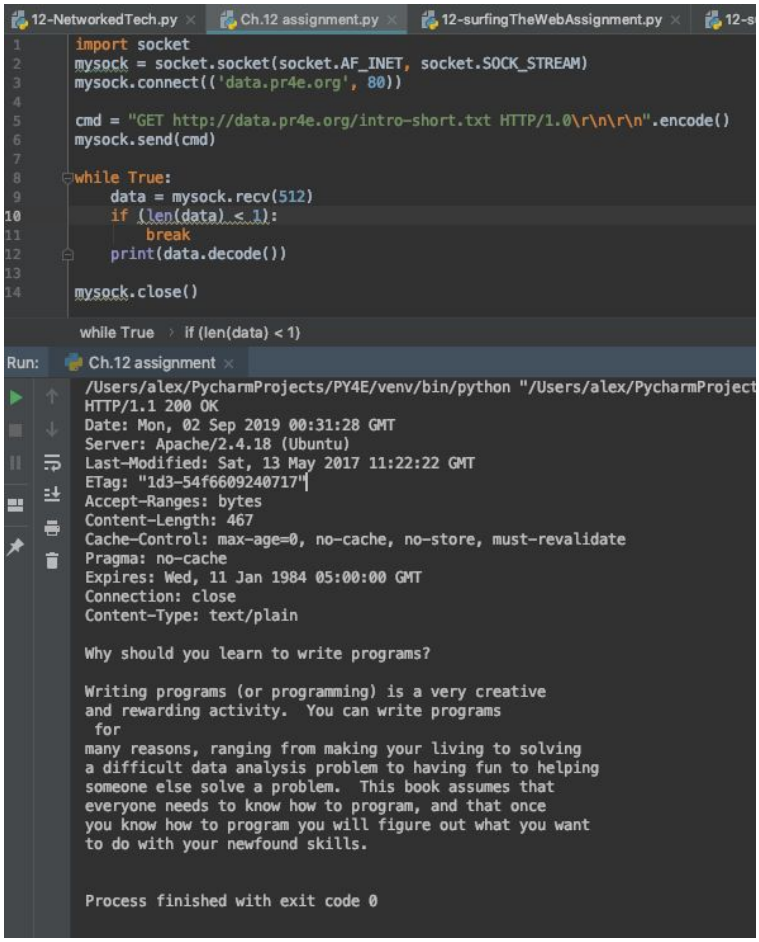
Sample lecture code with output from the web page in the console terminal:



Prompt for the assignment with required information entered:



Code to receive information for the assignment, information needed is in the console terminal:



D.3: Week 4 - Scraping HTML Data with Beautifulsoup: The Python file, urllink2.py, is provided in the Sample code folder that was downloaded at the beginning of the specialization as a reference. Use of **urllib** to read the html from the data file link entered by the user (html = request.urlopen(input("Enter - ")).read()), and use of Beautifulsoup to parse the data and extract the numbers (sopa = BeautifulSoup(html, "html.parser)). Computing the sum of numbers in the file with a for-loop.

Prompt for the assignment:

Welcome Alex Huerta from Using Python to Access Web Data

Your current grade on this assignment is: 100%

Scraping Numbers from HTML using BeautifulSoup In this assignment you will write a Python program similar to http://www.py4e.com/code3/urllink2.py. The program will use **urllib** to read the HTML from the data files below, and parse the data, extracting numbers and compute the sum of the numbers in the file.

We provide two files for this assignment. One is a sample file where we give you the sum for your testing and the other is the actual data you need to process for the assignment.

- Sample data: http://py4e-data.dr-chuck.net/comments_42.html (Sum=2553)
- Actual data: http://py4e-data.dr-chuck.net/comments_189370.html (Sum ends with 20)

You do not need to save these files to your folder since your program will read the data directly from the URL. **Note:** Each student will have a distinct data url for the assignment - so only use your own data url for analysis.

Data Format

The file is a table of names and comment counts. You can ignore most of the data in the file except for lines like the following:

```
<tr><td>Modu</td><td><span class="comments">90</span></td></tr>
<tr><td>Kenzie</td><td><span class="comments">88</span></td></tr>
<tr><td>Hubert</td><td><span class="comments">87</span></td></tr>
```

You are to find all the tags in the file and pull out the numbers from the tag and sum the numbers.

Look at the sample code provided. It shows how to find all of a certain kind of tag, loop through the tags and extract the various aspects of the tags.

```
...
# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
   # Look at the parts of a tag
   print 'TAG:',tag
   print 'URL:',tag.get('href', None)
   print 'Contents:',tag.contents[0]
   print 'Attrs:',tag.attrs
```

You need to adjust this code to look for **span** tags and pull out the text content of the span tag, convert them to integers and add them up to complete the assignment.

Sample Execution

```
$ python3 solution.py
Enter - http://py4e-data.dr-chuck.net/comments_42.html
Count 50
Sum 2...
```

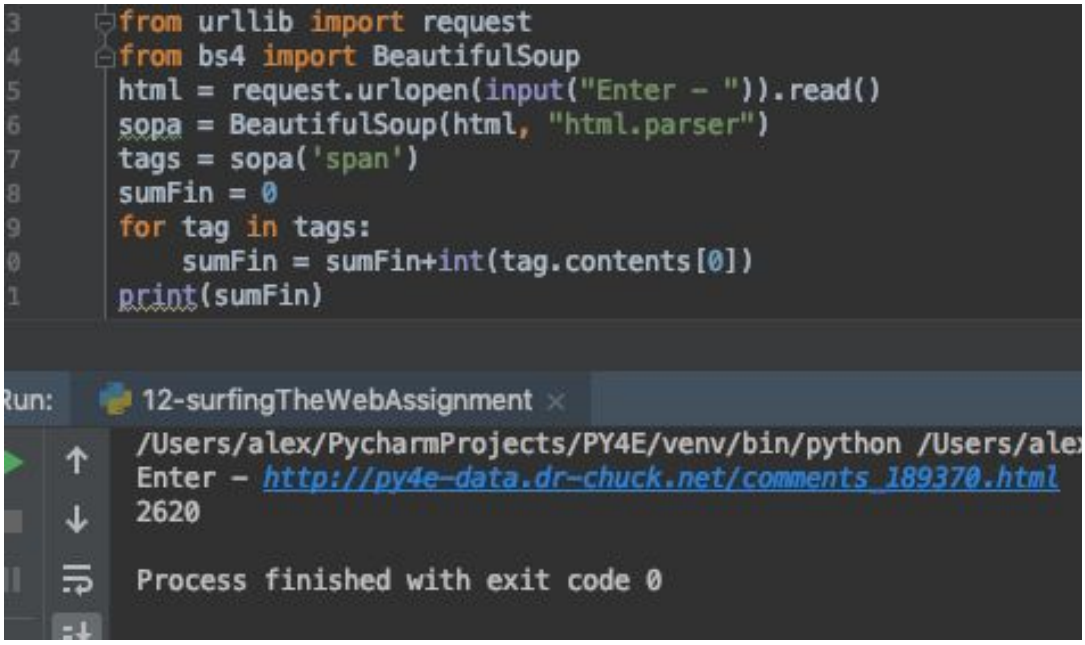
Turning in the Assignment

Enter the sum from the actual data and your Python code below:

Sum: 2620 (ends with 20) Submit Assignment

Python code:

Modified code to look for span tags and get the total amount of numbers in the those span tags, derived from urllink2.py. Answer in console terminal:



Week 4 - Following Links in HTML using BeautifulSoup: The Python file, urllinks.py, is provided in the Sample code folder that was downloaded at the beginning of the specialization as a reference. Use of **urllib** to read the HTML from the data file link entered (“html” variable). Use of BeautifulSoup to parse and extract the “href=” values from the anchor tags (“sopa” variable). For-loop to scan for a tag that is in a particular position (“pos” variable) relative to the first name in the list, follow that link and repeat the process a number of times (“cnt” variable) and report the last name you find.

Prompt for the assignment:

Welcome Alex Huerta from Using Python to Access Web Data

Your current grade on this assignment is: 100%

Following Links in Python

In this assignment you will write a Python program that expands on http://www.py4e.com/code3/urllinks.py. The program will use **urllib** to read the HTML from the data files below, extract the href= vaues from the anchor tags, scan for a tag that is in a particular position relative to the first name in the list, follow that link and repeat the process a number of times and report the last name you find.

We provide two files for this assignment. One is a sample file where we give you the name for your testing and the other is the actual data you need to process for the assignment

- Sample problem: Start at http://py4e-data.dr-chuck.net/known_by_Fikret.html
  Find the link at position **3** (the first name is 1). Follow that link. Repeat this process **4** times. The answer is the last name that you retrieve.
  Sequence of names: Fikret Montgomery Mhairade Butchi Anayah
  Last name in sequence: Anayah
- Actual problem: Start at: http://py4e-data.dr-chuck.net/known_by_Alexx.html
  Find the link at position **18** (the first name is 1). Follow that link. Repeat this process **7** times. The answer is the last name that you retrieve.
  Hint: The first character of the name of the last page that you will load is: M

Strategy

The web pages tweak the height between the links and hide the page after a few seconds to make it difficult for you to do the assignment without writing a Python program. But frankly with a little effort and patience you can overcome these attempts to make it a little harder to complete the assignment without writing a Python program. But that is not the point. The point is to write a clever Python program to solve the program.

Sample execution

Here is a sample execution of a solution:

```
$ python3 solution.py
Enter URL: http://py4e-data.dr-chuck.net/known_by_Fikret.html
Enter count: 4
Enter position: 3
Retrieving: http://py4e-data.dr-chuck.net/known_by_Fikret.html
Retrieving: http://py4e-data.dr-chuck.net/known_by_Montgomery.html
Retrieving: http://py4e-data.dr-chuck.net/known_by_Mhairade.html
Retrieving: http://py4e-data.dr-chuck.net/known_by_Butchi.html
Retrieving: http://py4e-data.dr-chuck.net/known_by_Anayah.html
```

The answer to the assignment for this execution is "Anayah".

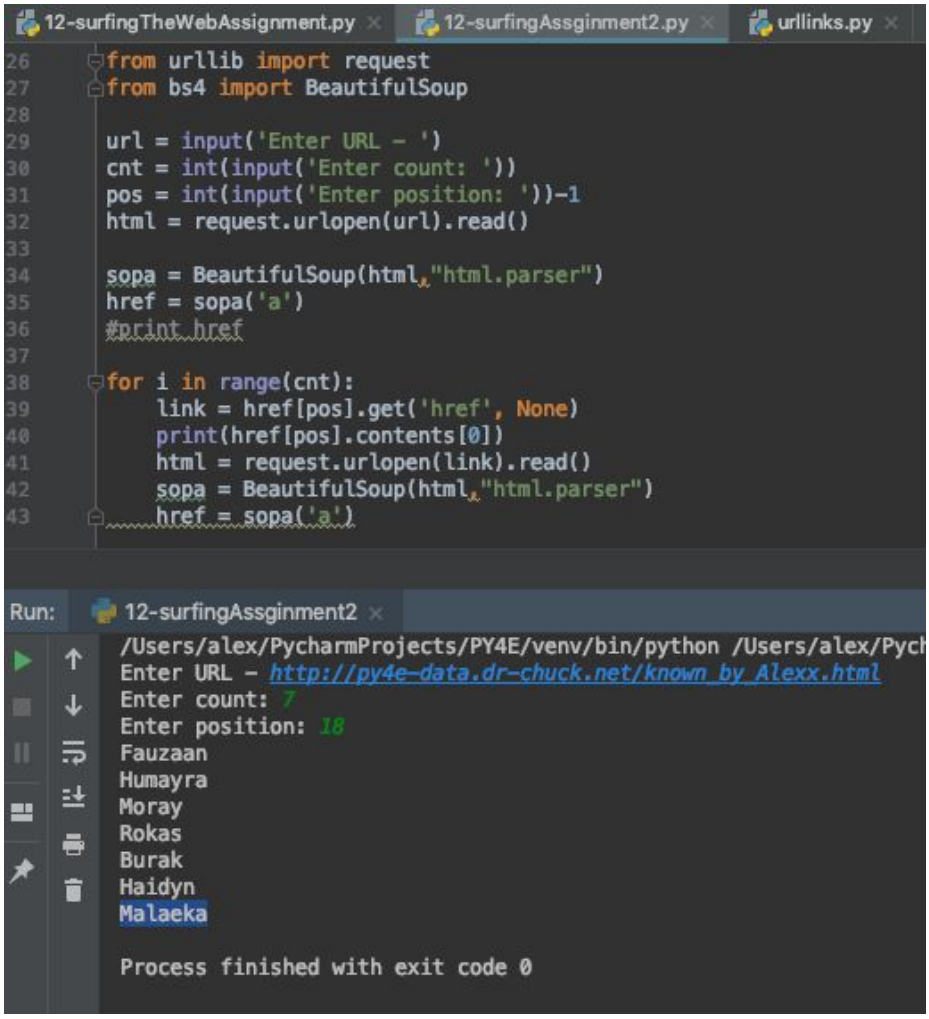
Turning in the Assignment

Enter the last name retrieved and your Python code below:

Name: Malaeka (name starts with M) Submit Assignment

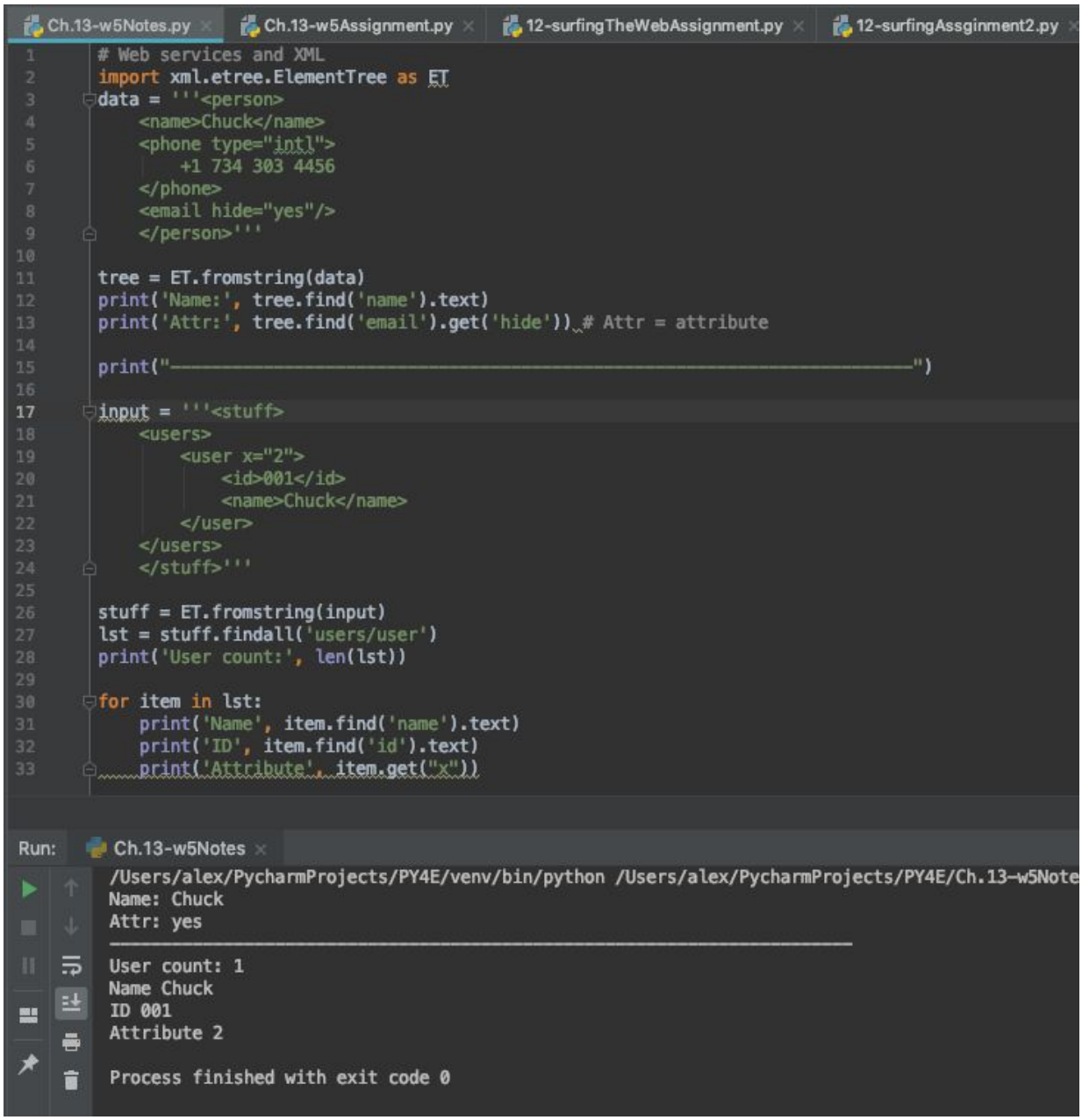
Python code:

Expanded version of urllinks.py code to retrieve the last names from the tags. Output in the console terminal:



D.4: Week 5 - Extracting Data from XML Assignment: Included my notes from the XML lecture so you can visualize an XML tree and get a better understanding of how XML data works with Python. The assignment requires the use of **urllib** to read the XML data from the URL entered (exmls = request.urlopen(address).read()), then uses the XML element tree to parse and extract the comment counts from the data (tree = ET.fromstring(exmls)). The program finds the counts by using the findall() function and sums them up with the for-loop.

My notes, displays XML data (XML tree) interacting with Python, console terminal shows output:



Prompt for the assignment:

Welcome Alex Huerta from Using Python to Access Web Data

Your current grade on this assignment is: 100%

Extracting Data from XML

In this assignment you will write a Python program somewhat similar to http://www.py4e.com/code3/geoxml.py. The program will prompt for a URL, read the XML data from that URL using **urllib** and then parse and extract the comment counts from the XML data, compute the sum of the numbers in the file.

We provide two files for this assignment. One is a sample file where we give you the sum for your testing and the other is the actual data you need to process for the assignment.

- Sample data: http://py4e-data.dr-chuck.net/comments_42.xml (Sum=2553)
- Actual data: http://py4e-data.dr-chuck.net/comments_189372.xml (Sum ends with 87)

You do not need to save these files to your folder since your program will read the data directly from the URL. **Note:** Each student will have a distinct data url for the assignment - so only use your own data url for analysis.

Data Format and Approach

The data consists of a number of names and comment counts in XML as follows:

```
<comment>
  <name>Matthias</name>
  <count>97</count>
</comment>
```

You are to look through all the <comment> tags and find the <count> values sum the numbers. The closest sample code that shows how to parse XML is geoxml.py. But since the nesting of the elements in our data is different than the data we are parsing in that sample code you will have to make real changes to the code.

To make the code a little simpler, you can use an XPath selector string to look through the entire tree of XML for any tag named 'count' with the following line of code:

```
counts = tree.findall('.//count')
```

Take a look at the Python ElementTree documentation and look for the supported XPath syntax for details. You could also work from the top of the XML down to the comments node and then loop through the child nodes of the comments node.

Sample Execution

```
$ python3 solution.py
Enter location: http://py4e-data.dr-chuck.net/comments_42.xml
Retrieving http://py4e-data.dr-chuck.net/comments_42.xml
Retrieved 4189 characters
Count: 50
Sum: 2...
```

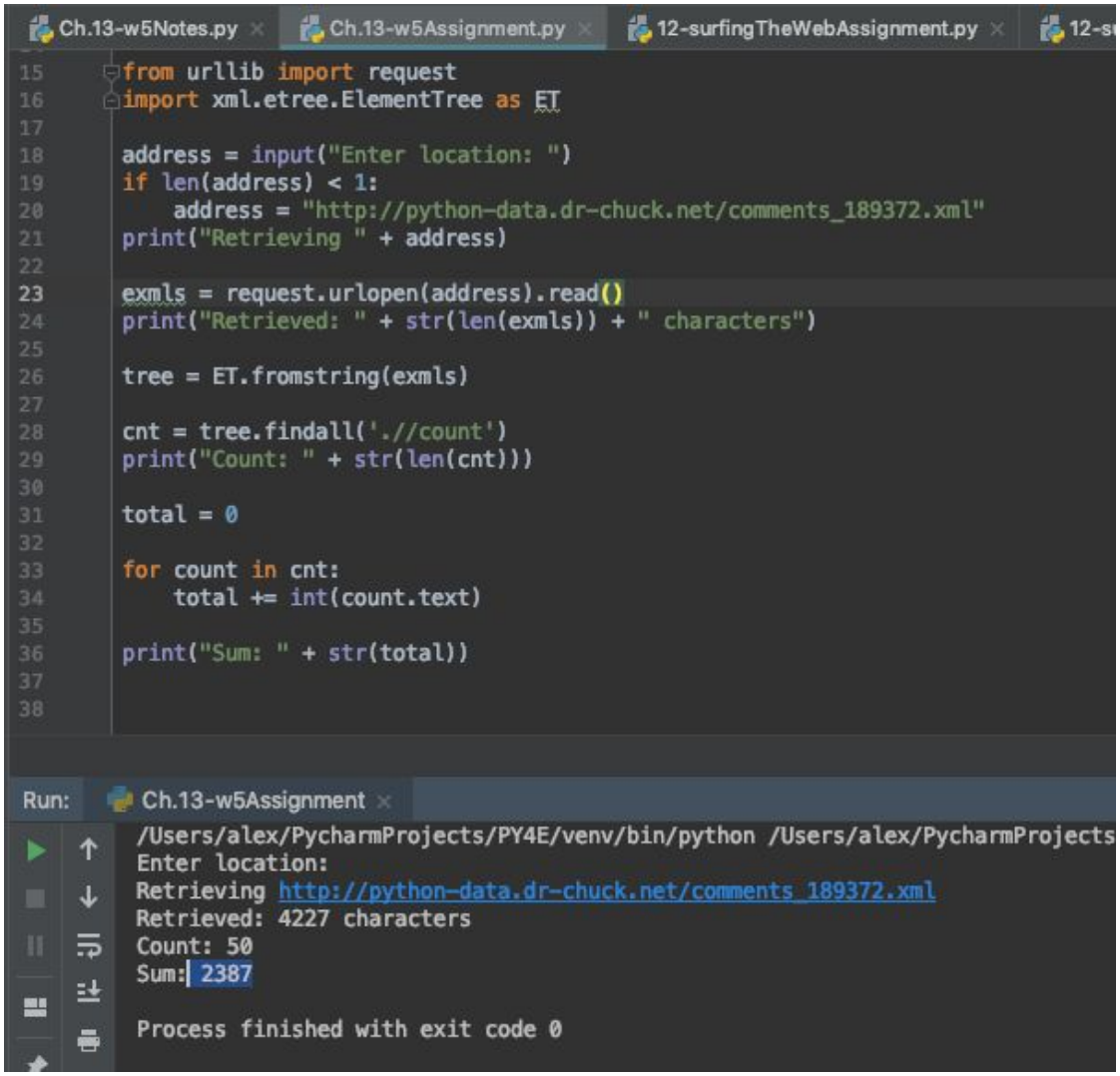
Turning in the Assignment

Enter the sum from the actual data and your Python code below:

Sum: 2387 (ends with 87) Submit Assignment

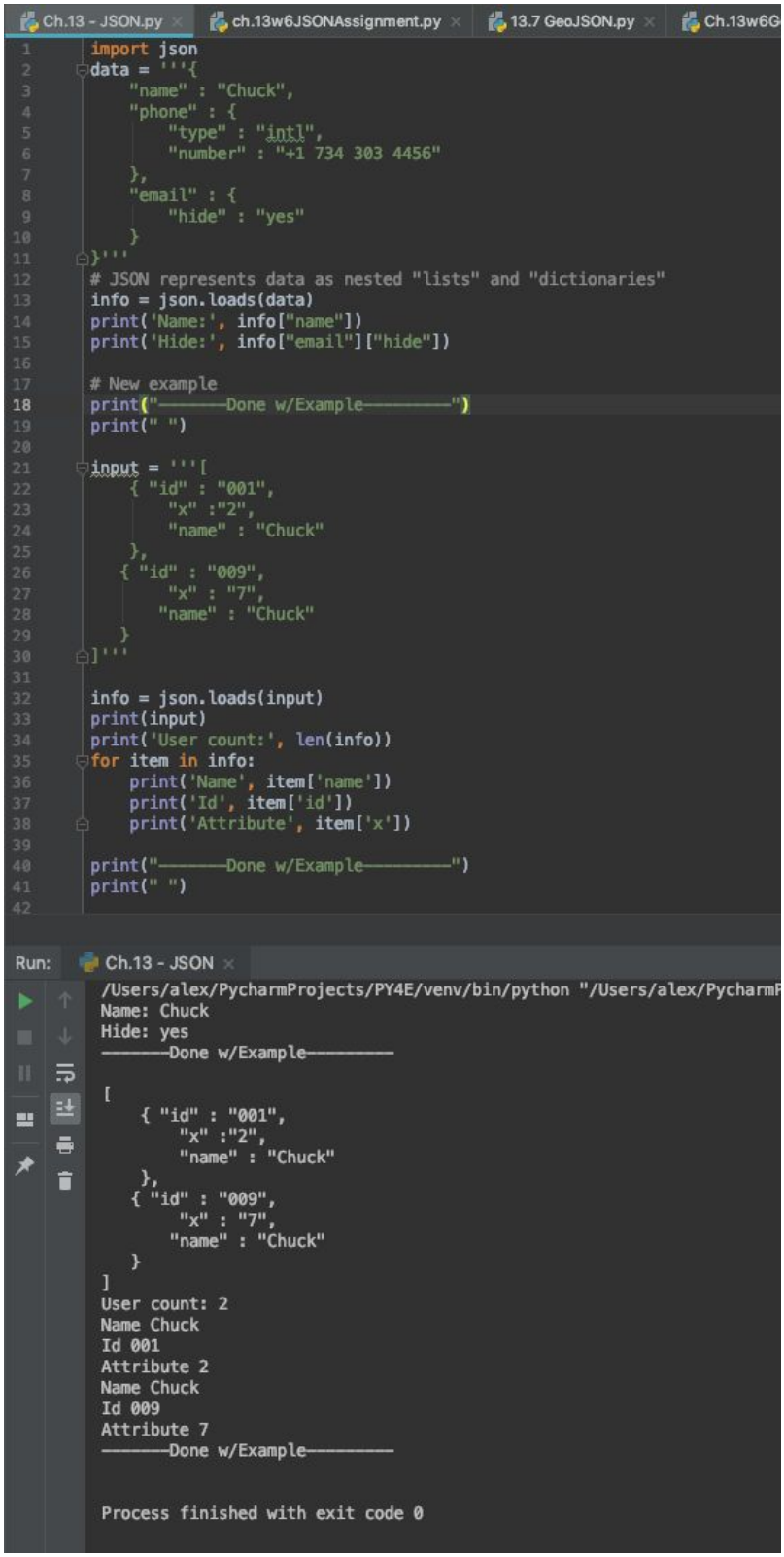
Python code:

Code for assignment that receives the sum of the counts, info needed is in the console terminal:



D.5: Week 6 - Extracting Data from JSON Assignment: Included my notes from the JSON lecture to help you visualize the JSON architecture and see how Python interacts with JSON and pulls data. The assignment will require you to read the JSON data from the URL entered (info = addy.read()) using **urllib**. Then you will need to use JSON to parse and extract comment counts from the JSON data (data = json.loads(info) and data = data[“comments”]). Finally we use the for-loop to sum all of the counts from the file.

My JSON notes from lecture, output from the code is shown in the console terminal:



```
import json
data = '''{
    "name" : "Chuck",
    "phone" : {
        "type" : "intl",
        "number" : "+1 734 303 4456"
    },
    "email" : {
        "hide" : "yes"
    }
}'''
# JSON represents data as nested "lists" and "dictionaries"
info = json.loads(data)
print('Name:', info["name"])
print('Hide:', info["email"]["hide"])

# New example
print("--------Done w/Example---------")
print(" ")

input = '''[
    { "id" : "001",
        "x" :"2",
        "name" : "Chuck"
    },
   { "id" : "009",
        "x" : "7",
       "name" : "Chuck"
   }
]'''

info = json.loads(input)
print(input)
print('User count:', len(info))
for item in info:
    print('Name', item['name'])
    print('Id', item['id'])
    print('Attribute', item['x'])

print("--------Done w/Example---------")
print(" ")
```

```
Run: Ch.13 - JSON
/Users/alex/PycharmProjects/PY4E/venv/bin/python "/Users/alex/PycharmP
Name: Chuck
Hide: yes
--------Done w/Example---------

[
    { "id" : "001",
        "x" :"2",
        "name" : "Chuck"
    },
   { "id" : "009",
        "x" : "7",
       "name" : "Chuck"
   }
]
User count: 2
Name Chuck
Id 001
Attribute 2
Name Chuck
Id 009
Attribute 7
--------Done w/Example---------


Process finished with exit code 0
```

Prompt for the assignment:

Welcome Alex Huerta from Using Python to Access Web Data

Your current grade on this assignment is: 100%

Extracting Data from JSON

In this assignment you will write a Python program somewhat similar to http://www.py4e.com/code3/json2.py. The program will prompt for a URL, read the JSON data from that URL using **urllib** and then parse and extract the comment counts from the JSON data, compute the sum of the numbers in the file and enter the sum below:

We provide two files for this assignment. One is a sample file where we give you the sum for your testing and the other is the actual data you need to process for the assignment.

- Sample data: http://py4e-data.dr-chuck.net/comments_42.json (Sum=2553)
- Actual data: http://py4e-data.dr-chuck.net/comments_189373.json (Sum ends with 58)

You do not need to save these files to your folder since your program will read the data directly from the URL. **Note:** Each student will have a distinct data url for the assignment - so only use your own data url for analysis.

Data Format

The data consists of a number of names and comment counts in JSON as follows:

```
{
  comments: [
    {
      name: "Matthias"
      count: 97
    },
    {
      name: "Geomer"
      count: 97
    }
    ...
  ]
}
```

The closest sample code that shows how to parse JSON and extract a list is json2.py. You might also want to look at geoxml.py to see how to prompt for a URL and retrieve data from a URL.

Sample Execution

```
$ python3 solution.py
Enter location: http://py4e-data.dr-chuck.net/comments_42.json
Retrieving http://py4e-data.dr-chuck.net/comments_42.json
Retrieved 2733 characters
Count: 50
Sum: 2...
```

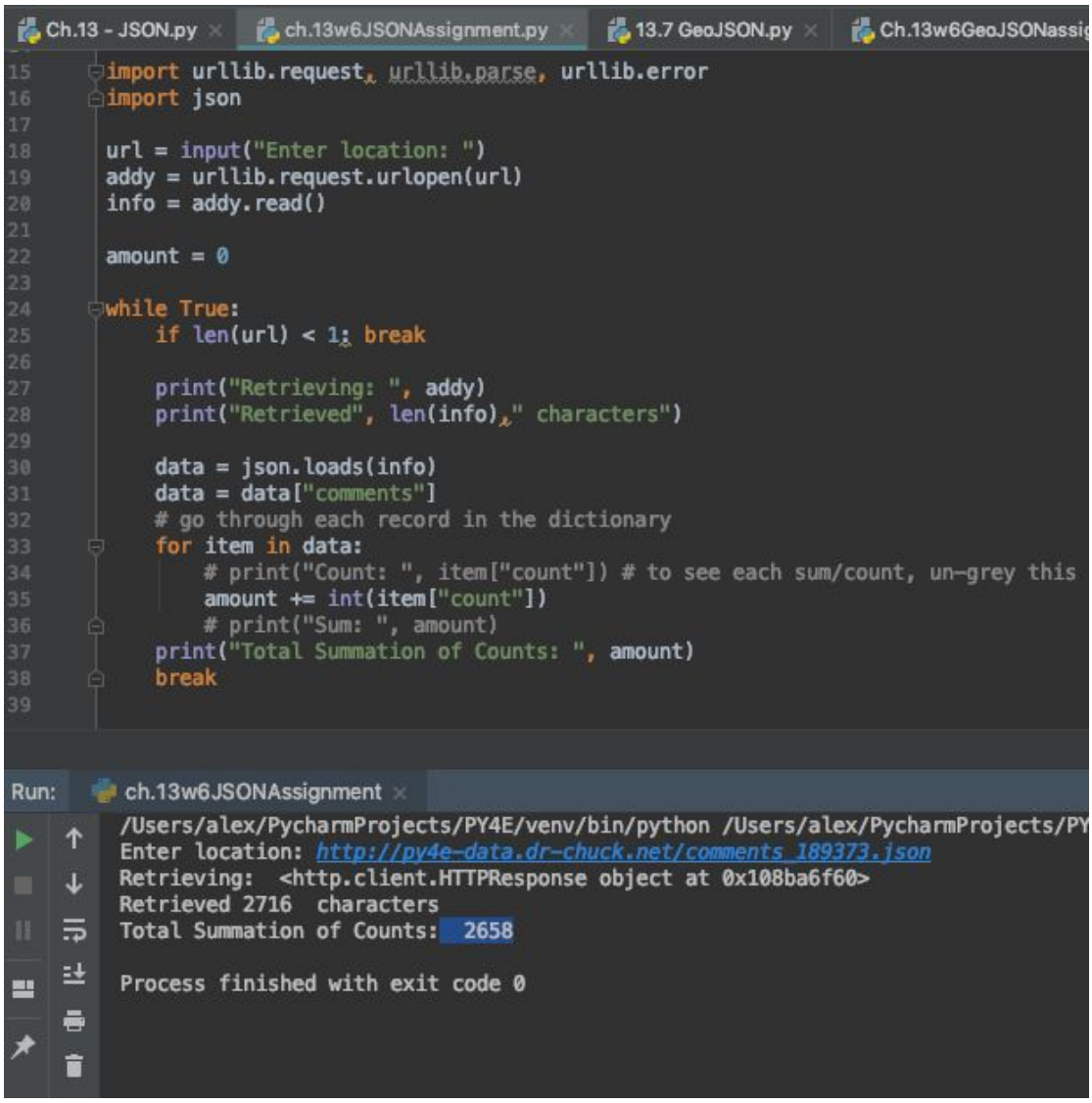
Turning in the Assignment

Enter the sum from the actual data and your Python code below:

Sum: 2658 (ends with 58) Submit Assignment

Python code:

Code for assignment that calculates the sum of all the numbers in the JSON data, sum is provided in the console terminal:



Week 6 - Using the GeoJSON API Assignment: The Python file, geojson.py, is provided in the Sample code folder that was downloaded at the beginning of the specialization as a reference. Use of the API endpoint (http.//py4e-data.dr.chuck.net/json?) which has a static subset of Google's GeoCode data (lines 34-48 in the code for the assignment). The program then contacts a web service and retrieves JSON data for the web service and parses it (lines 57-62 in the code for the assignment), then receives the first "place_id" from the JSON data (js = json.loads(data) and print("Place id: ", js["results"] [0] ["place_id"]).

Prompt for the assignment:

Welcome Alex Huerta from Using Python to Access Web Data

Your current grade on this assignment is: 100%

Calling a JSON API

In this assignment you will write a Python program somewhat similar to http://www.py4e.com/code3/geojson.py. The program will prompt for a location, contact a web service and retrieve JSON for the web service and parse that data, and retrieve the first **place_id** from the JSON. A place ID is a textual identifier that uniquely identifies a place as within Google Maps.

API End Points

To complete this assignment, you should use this API endpoint that has a static subset of the Google Data:

```
http://py4e-data.dr-chuck.net/json?
```

This API uses the same parameter (address) as the Google API. This API also has no rate limit so you can test as often as you like. If you visit the URL with no parameters, you get "No address..." response.

To call the API, you need to provide the address that you are requesting as the **address=** parameter that is properly URL encoded using the **urllib.parse.urlencode()** function as shown in http://www.py4e.com/code3/geojson.py

Test Data / Sample Execution

You can test to see if your program is working with a location of "South Federal University" which will have a **place_id** of "ChIJLRafow6StocRyuUGlcx9_m8".

```
$ python3 solution.py
Enter location: South Federal University
Retrieving http://...
Retrieved 2418 characters
Place id ChIJLRafow6StocRyuUGlcx9_m8
```

Turn In

Please run your program to find the **place_id** for this location:

```
IU
```

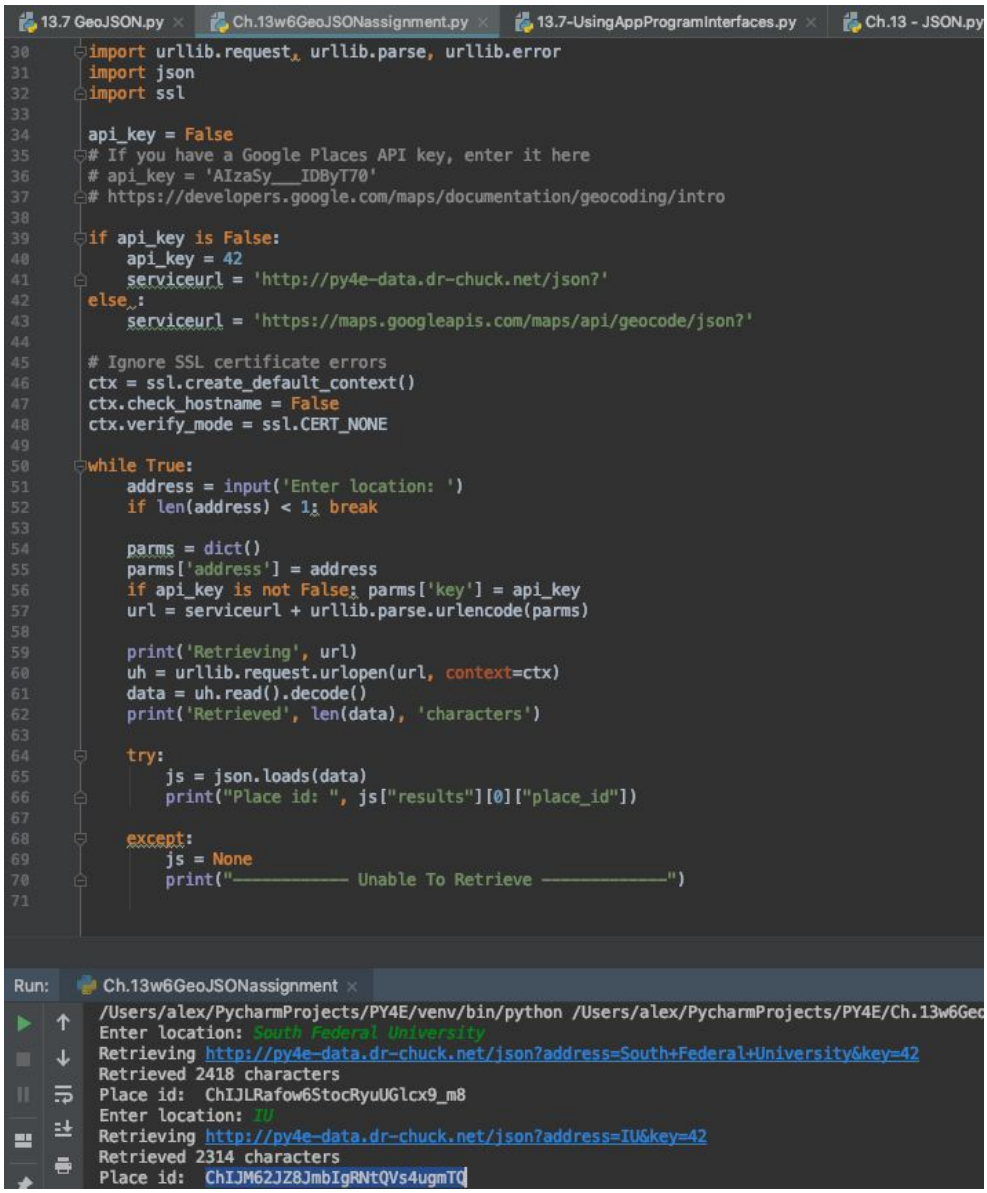
Make sure to enter the name and case exactly as above and enter the **place_id** and your Python code below. Hint: The first seven characters of the **place_id** are "ChIJM62 ..."

Make sure to retreive the data from the URL specified above and **not** the normal Google API. Your program should work with the Google API - but the **place_id** may not match for this assignment.

place_id: ChIJM62JZ8JmbIgRNtQVs4ugmTQ Submit Assignment

Python code:

Modified code to retrieve the first “place_id” from the “IU” location, derived from “geojson.py”. Outputs from test location (South Federal University) and actual location (IU) displayed in console terminal:



E. Course 4:

E.1: Week 1 - Notes on Object Oriented programming (Objects, Classes, Inheritance), from YouTube Python tutorial.

Here we are simply defining (creating) the “Chef” class and assigning it 3 functions/methods: make_chicken, make_salad, make_special_dish.

```python
class Chef:

    def make_chicken(self):
        print("The chef makes a chicken")

    def make_salad(self):
        print("The chef makes a salad")

    def make_special_dish(self):
        print("The chef makes bbq ribs")
```

Next, to introduce inheritance, we are creating another class (Chinese_Chef) and we are importing the Chef class from the Chef.py file and passing it to the Chinese_Chef class. This means that the Chinese_Chef class will have all the functions from the Chef class, plus the new ones (make_fried_rice) and by defining the make_special_dish again this will overwrite what the function does, for the Chinese_Chef only.

```python
from Chef import Chef

class Chinese_Chef(Chef):

    def make_fried_rice(self):
        print("The chef makes fried rice")

    def make_special_dish(self):
        print("The chef makes orange chicken ")
```

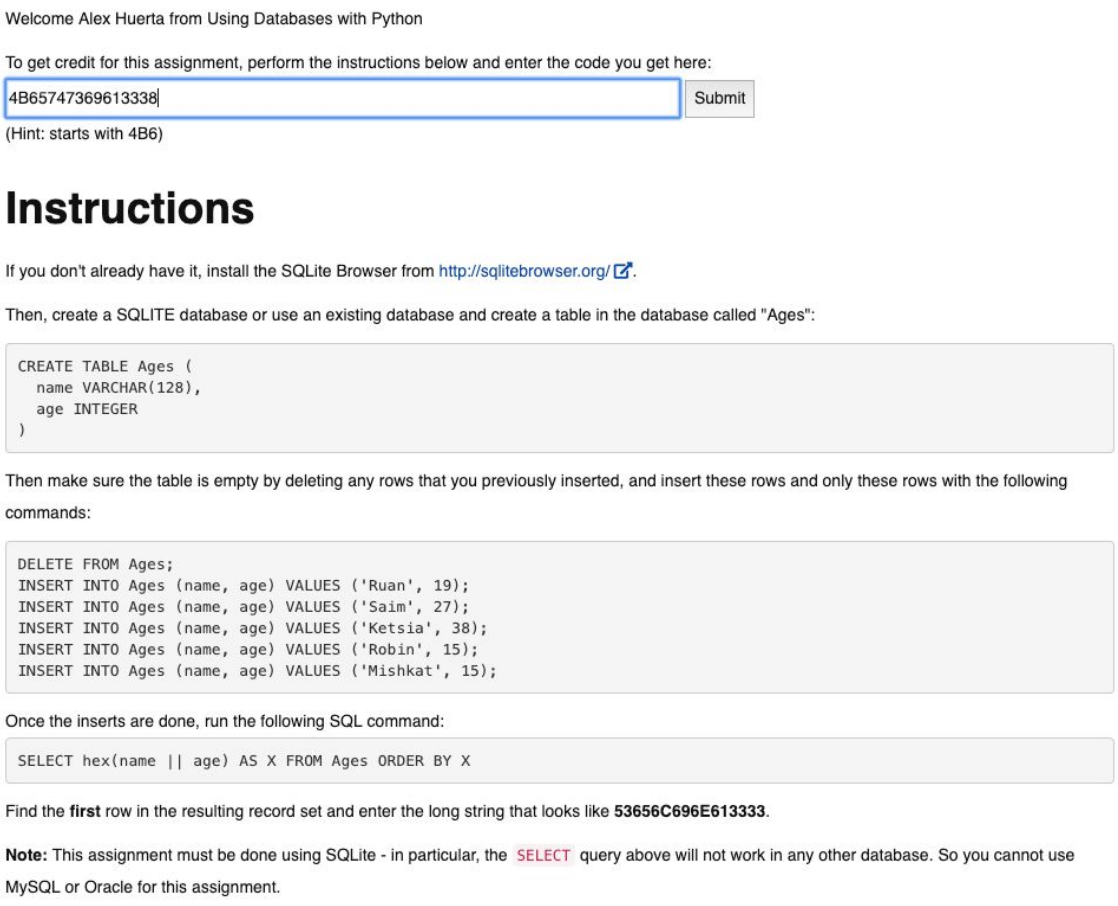
To show how inheritance works, in a separate file we'll import both classes and call on functions from both classes. Notice how the Chinese_Chef class is able to run the make_chicken function, and the make_special_dish function is different from Chef.

```python
from Chef import Chef
from Chinese_Chef import Chinese_Chef

myChef = Chef()
myChef.make_chicken()
myChef.make_special_dish()
print("_____")
myChinese_Chef = Chinese_Chef()
myChinese_Chef.make_chicken()
myChinese_Chef.make_special_dish()
```

```
Inheritance
/Users/alex/PycharmProjects/untitled/venv/bin/pyt
The chef makes a chicken
The chef makes bbq ribs
_____
The chef makes a chicken
The chef makes orange chicken

Process finished with exit code 0
```

E.2: Week 2 - Our First Database Assignment: This is solely executed in SQLite, no Python. After copying and pasting the Create statement and the Delete/Insert statement from the prompt into the “Execute SQL” option in SQLite we are left with an “Ages” table. Then after running the Select statement from the prompt in SQLite we are provided with the information needed to answer the prompt.

Prompt for the assignment:



Welcome Alex Huerta from Using Databases with Python

To get credit for this assignment, perform the instructions below and enter the code you get here:

4B65747369613338 Submit

(Hint: starts with 4B6)

Instructions

If you don't already have it, install the SQLite Browser from http://sqlitebrowser.org/.

Then, create a SQLITE database or use an existing database and create a table in the database called "Ages":

```
CREATE TABLE Ages (
  name VARCHAR(128),
  age INTEGER
)
```

Then make sure the table is empty by deleting any rows that you previously inserted, and insert these rows and only these rows with the following commands:

```
DELETE FROM Ages;
INSERT INTO Ages (name, age) VALUES ('Ruan', 19);
INSERT INTO Ages (name, age) VALUES ('Saim', 27);
INSERT INTO Ages (name, age) VALUES ('Ketsia', 38);
INSERT INTO Ages (name, age) VALUES ('Robin', 15);
INSERT INTO Ages (name, age) VALUES ('Mishkat', 15);
```

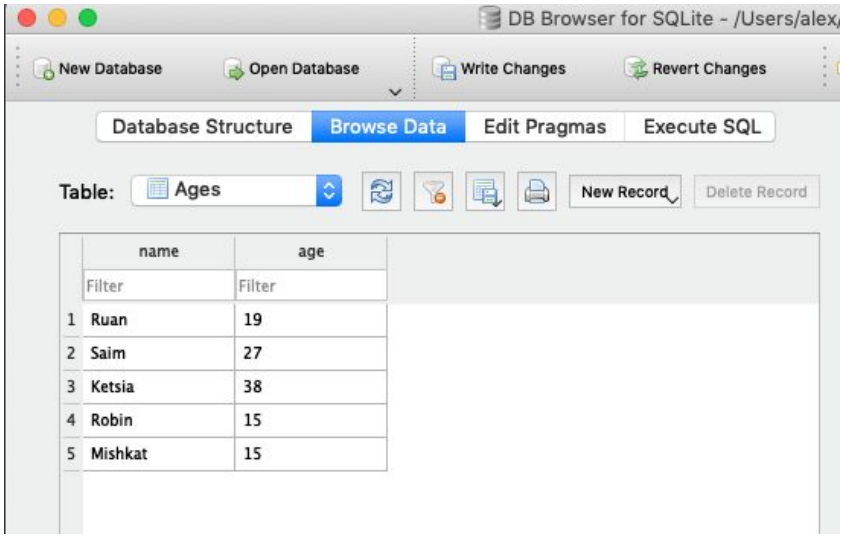
Once the inserts are done, run the following SQL command:

```
SELECT hex(name || age) AS X FROM Ages ORDER BY X
```

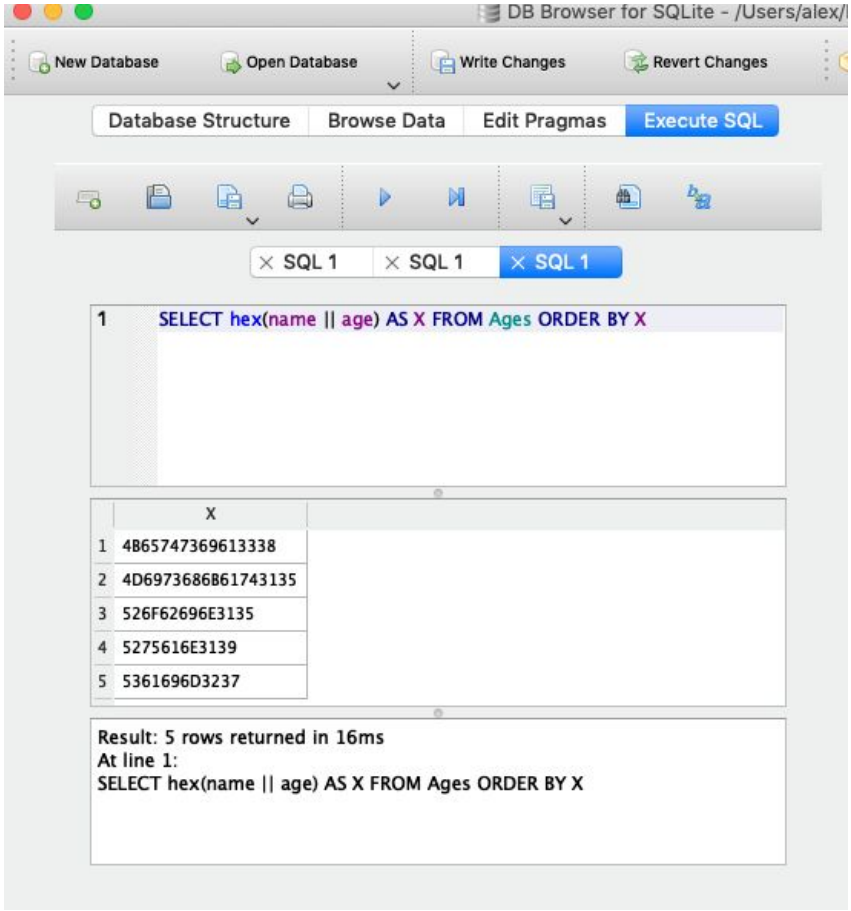
Find the **first** row in the resulting record set and enter the long string that looks like **53656C696E613333**.

Note: This assignment must be done using SQLite - in particular, the `SELECT` query above will not work in any other database. So you cannot use MySQL or Oracle for this assignment.

“Ages” table resulting from Create and Delete/Insert statements:



Output provided from the Select statement, first row is the data needed:



Week 2 - Counting Email in a Database Assignment: The Python file, emaildb.py, is provided in the Sample code folder that was downloaded at the beginning of the specialization as a reference. Use of Python and SQL to create a new database (conn = sqlite3.connect('CountingOrgs.sqlite')). Lines 16-17 create the table "Counts", lines 27-28 check if the received domains exist in the database. Use of the If-Else statement to insert the domains received from the mbox.txt file (lines 20-25). Lines 38-40 run the Select statement through a for-loop to print the results to the screen in Python. Submit the database that was created to complete the assignment.

Prompt for the assignment:

Welcome Alex Huerta from Using Databases with Python

To get credit for this assignment, perform the instructions below and upload your SQLite3 database here:

Choose File CountingOrgs.sqlite

(Must have a .sqlite suffix)

Hint: The top organizational count is 536.

Submit

You do not need to export or convert the database - simply upload the **.sqlite** file that your program creates. See the example code for the use of the **connect()** statement.

Counting Organizations

This application will read the mailbox data (mbox.txt) and count the number of email messages per organization (i.e. domain name of the email address) using a database with the following schema to maintain the counts.

```
CREATE TABLE Counts (org TEXT, count INTEGER)
```

When you have run the program on **mbox.txt** upload the resulting database file above for grading.

If you run the program multiple times in testing or with dfferent files, make sure to empty out the data before each run.

You can use this code as a starting point for your application: http://www.py4e.com/code3/emaildb.py.

The data file for this application is the same as in previous assignments: http://www.py4e.com/code3/mbox.txt.

Because the sample code is using an **UPDATE** statement and committing the results to the database as each record is read in the loop, it might take as long as a few minutes to process all the data. The commit insists on completely writing all the data to disk every time it is called.

The program can be speeded up greatly by moving the commit operation outside of the loop. In any database program, there is a balance between the number of operations you execute between commits and the importance of not losing the results of operations that have not yet been committed.

Code for the assignment that produces the sum of the domains in descending order:

```
import sqlite3

# Read the filename or if null use the standard name
name = input("Enter file: ")
if len(name) < 1: name = "mbox.txt"
handle = open(name)

templist = list()
count = 0

# Open database connection
conn = sqlite3.connect('CountingOrgs.sqlite')
cur = conn.cursor()

# Create table or delete its content
cur.execute('''CREATE TABLE IF NOT EXISTS Counts (org TEXT, count INTEGER)''')
cur.execute('''DELETE FROM Counts''')

# Find all emails in the From lines
for line in handle:
    line = line.rstrip()
    if not line.startswith('From '): continue
    # Get the domains
    words = line.split()
    domain = words[1].split("@")
    # Check if domain exist in the database
    cur.execute('''SELECT count FROM Counts where org = ?''', (domain[1],))
    count = cur.fetchone()
    # If not create an entry
    if count is None:
        cur.execute('''INSERT INTO Counts (org, count) VALUES (?,1)''', (domain[1],))
    else:
        # If it does update the record
        cur.execute('''UPDATE Counts SET count = count + 1  WHERE org = ?''', (domain[1],))

conn.commit()

table = 'SELECT org, count FROM Counts ORDER BY count DESC'
for line in cur.execute(table):
    print(str(line[0]), ":", str(line[1]))

conn.close()
```

```
Run:  CH15assignment2
/Users/alex/PycharmProjects/PY4E/venv/bin/python /Users/alex/PycharmProjects/PY4E/CH15assign
Enter file: mbox.txt
iupui.edu : 536
umich.edu : 491
indiana.edu : 178
caret.cam.ac.uk : 157
vt.edu : 110
uct.ac.za : 96
media.berkeley.edu : 56
ufp.pt : 28
gmail.com : 25
et.gatech.edu : 17
txstate.edu : 17
whitman.edu : 17
lancaster.ac.uk : 14
bu.edu : 14
stanford.edu : 12
unicon.net : 9
loi.nl : 9
rsmart.com : 8
ucdavis.edu : 1
fhda.edu : 1
utoronto.ca : 1

Process finished with exit code 0
```

E.3: Week 3 - “Multi-Table Database - Tracks” Assignment: Tracks.py and Library.xml are provided in the Sample code folder that was downloaded at the beginning of the specialization, as a reference. I have commented (#) notes on the modified version of the tracks.py file, to explain the code. The gist of it is, lines 15-42 is where we put the

Create statement provided in the prompt (creates the multiple tables and deletes their content), lines 86-109 is where we insert the data retrieved from reading the XML file into the tables, and lines 112-120 is where we put the Select statement (uses JOINS to pull data from multiple tables), provided in the prompt, into a variable, which we can then print out by using a for-loop. Submit the database that has been created (trackdb.sqlite) to complete the assignment.

Prompt for the assignment:

Welcome Alex Huerta from Using Databases with Python

Your current grade on this assignment is: 100%

To get credit for this assignment, perform the instructions below and upload your SQLite3 database here:

Choose File No file chosen

(Must have a .sqlite suffix)

Submit

You do not need to export or convert the database - simply upload the **.sqlite** file that your program creates. See the example code for the use of the **connect()** statement.

Musical Track Database

This application will read an iTunes export file in XML and produce a properly normalized database with this structure:

```
CREATE TABLE Artist (
    id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT UNIQUE
);

CREATE TABLE Genre (
    id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT UNIQUE
);

CREATE TABLE Album (
    id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    artist_id  INTEGER,
    title   TEXT UNIQUE
);

CREATE TABLE Track (
    id  INTEGER NOT NULL PRIMARY KEY
        AUTOINCREMENT UNIQUE,
    title TEXT  UNIQUE,
    album_id  INTEGER,
    genre_id  INTEGER,
    len INTEGER, rating INTEGER, count INTEGER
);
```

If you run the program multiple times in testing or with different files, make sure to empty out the data before each run.

You can use this code as a starting point for your application: http://www.py4e.com/code3/tracks.zip. The ZIP file contains the **Library.xml** file to be used for this assignment. You can export your own tracks from iTunes and create a database, but for the database that you turn in for this assignment, only use the **Library.xml** data that is provided.

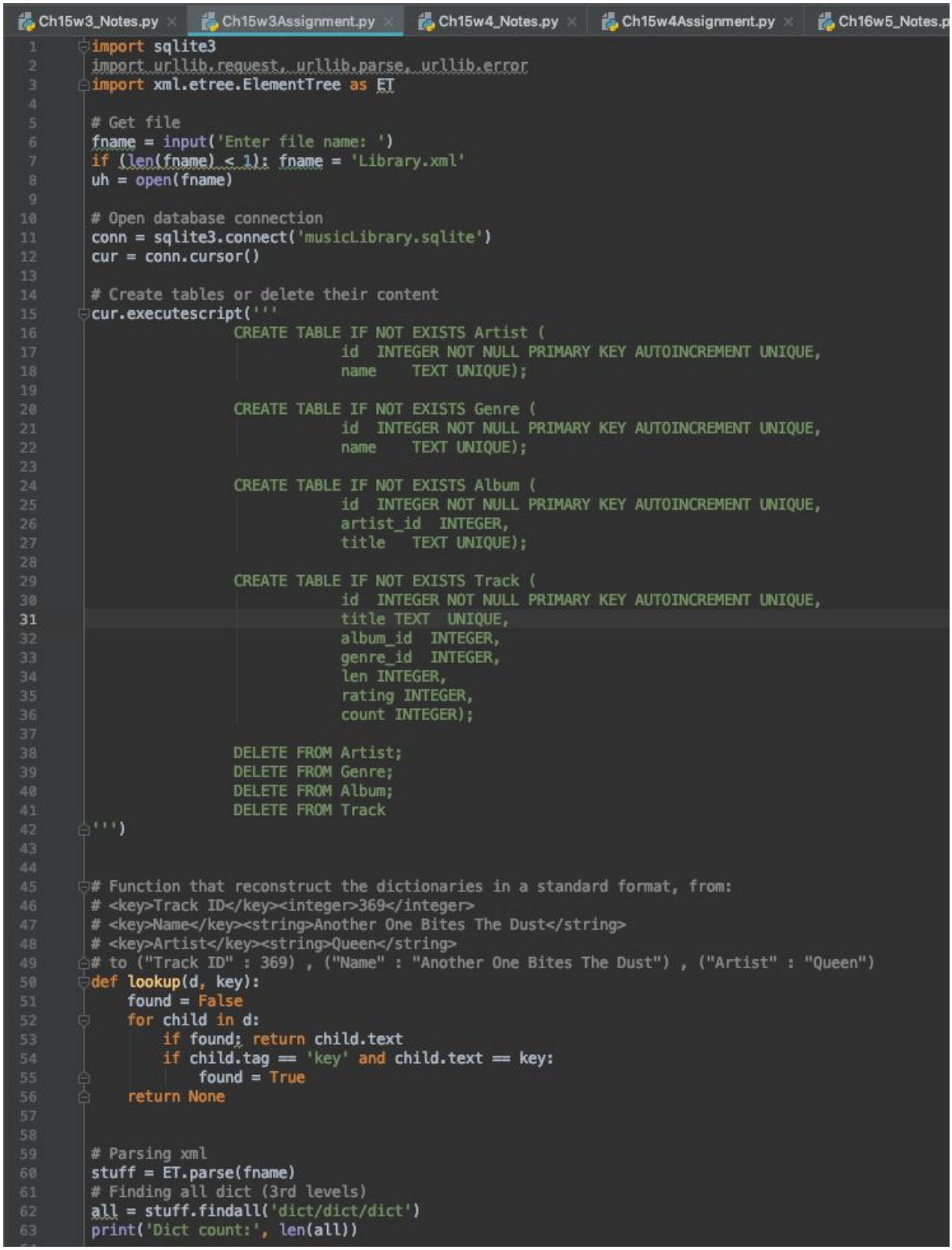
To grade this assignment, the program will run a query like this on your uploaded database and look for the data it expects to see:

```
SELECT Track.title, Artist.name, Album.title, Genre.name
    FROM Track JOIN Genre JOIN Album JOIN Artist
    ON Track.genre_id = Genre.ID and Track.album_id = Album.id
        AND Album.artist_id = Artist.id
    ORDER BY Artist.name LIMIT 3
```

The expected result of the modified query on your database is: (shown here as a simple HTML table with titles)

Track	Artist	Album	Genre
Chase the Ace	AC/DC	Who Made Who	Rock
D.T.	AC/DC	Who Made Who	Rock
For Those About To Rock (We Salute You)	AC/DC	Who Made Who	Rock

Modified code, derived from track.py, with the output (track names, band, album, and genre) in the console terminal:



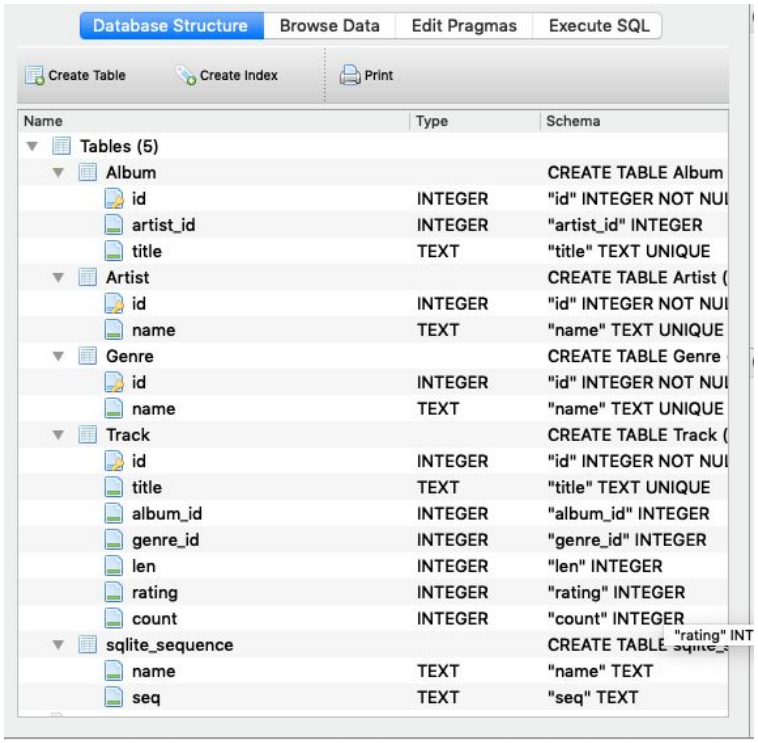
```
import sqlite3
import urllib.request, urllib.parse, urllib.error
import xml.etree.ElementTree as ET

# Get file
fname = input('Enter file name: ')
if (len(fname) < 1): fname = 'Library.xml'
uh = open(fname)

# Open database connection
conn = sqlite3.connect('musicLibrary.sqlite')
cur = conn.cursor()

# Create tables or delete their content
cur.executescript('''
                CREATE TABLE IF NOT EXISTS Artist (
                            id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                            name    TEXT UNIQUE);

                CREATE TABLE IF NOT EXISTS Genre (
                            id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                            name    TEXT UNIQUE);

                CREATE TABLE IF NOT EXISTS Album (
                            id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                            artist_id  INTEGER,
                            title   TEXT UNIQUE);

                CREATE TABLE IF NOT EXISTS Track (
                            id  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                            title TEXT  UNIQUE,
                            album_id  INTEGER,
                            genre_id  INTEGER,
                            len INTEGER,
                            rating INTEGER,
                            count INTEGER);

                DELETE FROM Artist;
                DELETE FROM Genre;
                DELETE FROM Album;
                DELETE FROM Track
''')


# Function that reconstruct the dictionaries in a standard format, from:
# <key>Track ID</key><integer>369</integer>
# <key>Name</key><string>Another One Bites The Dust</string>
# <key>Artist</key><string>Queen</string>
# to ("Track ID" : 369) , ("Name" : "Another One Bites The Dust") , ("Artist" : "Queen")
def lookup(d, key):
    found = False
    for child in d:
        if found: return child.text
        if child.tag == 'key' and child.text == key:
            found = True
    return None


# Parsing xml
stuff = ET.parse(fname)
# Finding all dict (3rd levels)
all = stuff.findall('dict/dict/dict')
print('Dict count:', len(all))
```

Code pt.2 :

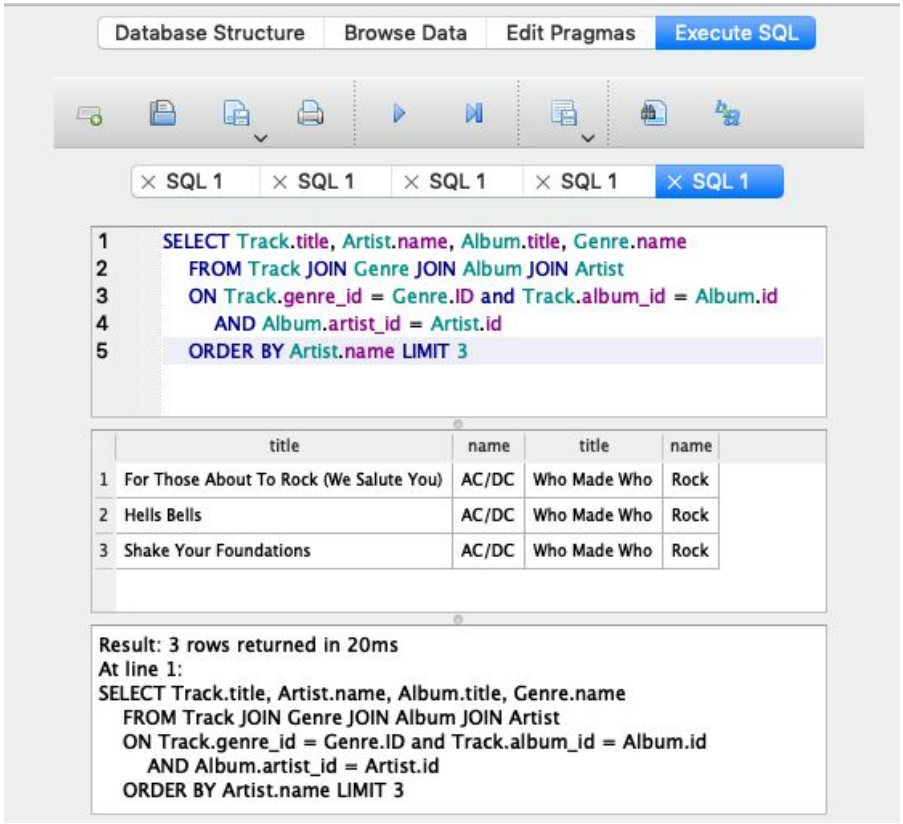
```
# For each dict (track) insert values in the database
for entry in all:

    # Cleaning up some incomplete records
    if (lookup(entry, 'Track ID') is None): continue
    if (lookup(entry, 'Genre') is None): continue

    # Collecting all fields from the track
    name = lookup(entry, 'Name')
    artist = lookup(entry, 'Artist')
    album = lookup(entry, 'Album')
    count = lookup(entry, 'Play Count')
    rating = lookup(entry, 'Rating')
    length = lookup(entry, 'Total Time')
    genre = lookup(entry, 'Genre')

    if name is None or artist is None or album is None: continue

    # print(name, artist, album, count, rating, length, genre)

    # Insert or ignore if already exist this record in Artist
    cur.execute('''INSERT OR IGNORE INTO Artist (name)
        VALUES ( ? )''', (artist,))
    cur.execute('SELECT id FROM Artist WHERE name = ? ', (artist,))
    artist_id = cur.fetchone()[0]

    # Insert or ignore if already exist this record in Album
    cur.execute('''INSERT OR IGNORE INTO Album (title, artist_id)
        VALUES ( ?, ? )''', (album, artist_id))
    cur.execute('SELECT id FROM Album WHERE title = ? ', (album,))
    album_id = cur.fetchone()[0]

    # Insert or ignore if already exist this record in Genre
    cur.execute('''INSERT OR IGNORE INTO Genre (name)
        VALUES ( ? )''', (genre,))
    cur.execute('SELECT id FROM Genre WHERE name = ? ', (genre,))
    genre_id = cur.fetchone()[0]

    # Insert or update values in Track
    cur.execute('''INSERT OR REPLACE INTO Track
        (title, album_id, len, rating, count, genre_id)
        VALUES ( ?, ?, ?, ?, ?, ? )''',
                (name, album_id, length, rating, count, genre_id))

conn.commit()

# Result query
table = '''SELECT Track.title, Artist.name, Album.title, Genre.name
            FROM Track JOIN Genre JOIN Album JOIN Artist
            ON Track.genre_id = Genre.ID and Track.album_id = Album.id
            AND Album.artist_id = Artist.id
            ORDER BY Artist.name LIMIT 3'''
for line in cur.execute(table):
    print(str(line[0]), "|", str(line[1]), "|", str(line[2]), "|", str(line[3]))

conn.close()
```

```
Run:  Ch15w3Assignment
/Users/alex/PycharmProjects/PY4E/venv/bin/python /Users/alex/PycharmProjects/PY4E/Ch15w3
Enter file name:
Dict count: 404
For Those About To Rock (We Salute You) | AC/DC | Who Made Who | Rock
Hells Bells | AC/DC | Who Made Who | Rock
Shake Your Foundations | AC/DC | Who Made Who | Rock

Process finished with exit code 0
```

“musicLibrary.sqlite” database structure, resulting from the Create statement:

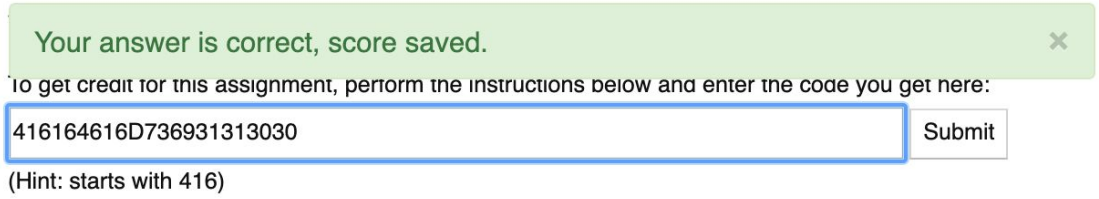


The Select statement from the prompt running in the SQLite browser:



E.4: Week 4 - “Many Students in Many Courses” Assignment: The python file, roster.py, is provided in the “Sample code” folder that was downloaded at the beginning of the specialization, as a reference. This assignment is fairly simple, all that needs to be done is to modify the code to include a “role” column in the SQL code (lines 44 and 79), as well as the Python code (lines 64, 66, 80). Then after running the modified python file to create the database, you run the Select statement (uses JOIN) provided in the prompt to get the data needed to complete the assignment.

Prompt for the assignment:



Instructions

This application will read roster data in JSON format, parse the file, and then produce an SQLite database that contains a User, Course, and Member table and populate the tables from the data file.

You can base your solution on this code: http://www.py4e.com/code3/roster/roster.py - this code is incomplete as you need to modify the program to store the **role** column in the **Member** table to complete the assignment.

Each student gets their own file for the assignment. Download this file and save it as `roster_data.json`. Move the downloaded file into the same folder as your `roster.py` program.

Once you have made the necessary changes to the program and it has been run successfully reading the above JSON data, run the following SQL command:

```
SELECT hex(User.name || Course.title || Member.role ) AS X FROM
    User JOIN Member JOIN Course
    ON User.id = Member.user_id AND Member.course_id = Course.id
    ORDER BY X
```

Find the **first** row in the resulting record set and enter the long string that looks like **53656C696E613333**.

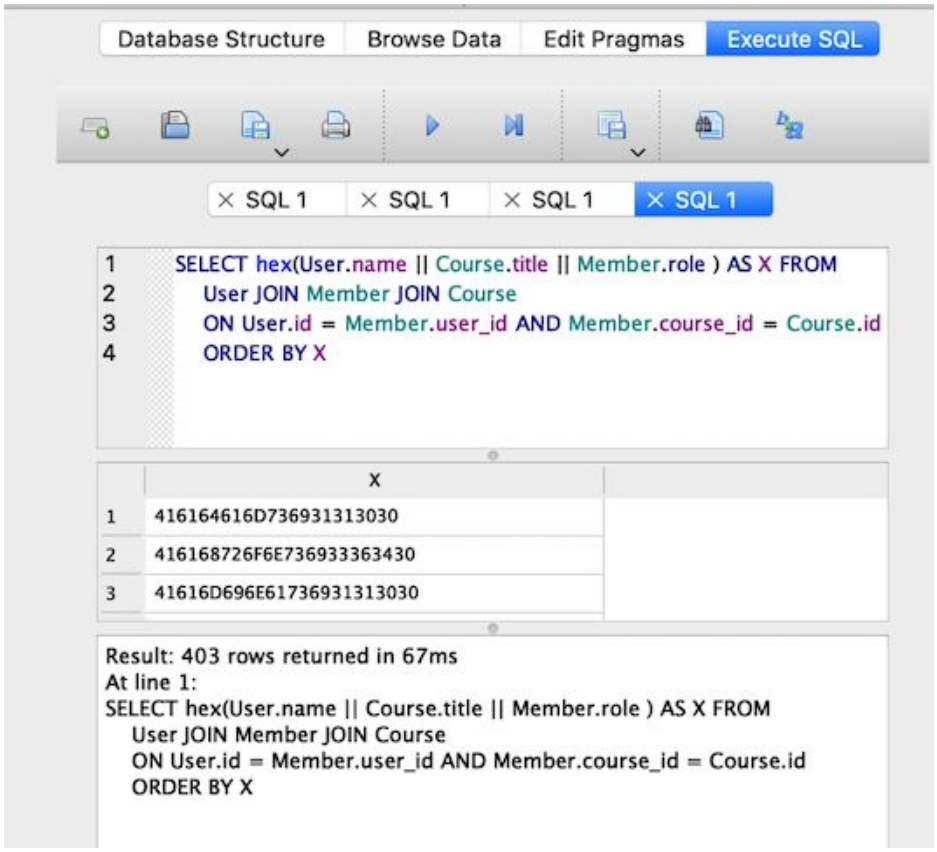
Modified version of roster.py code to include the “role column”:

```
import json
import sqlite3

conn = sqlite3.connect('rosterdb.sqlite')
cur = conn.cursor()

# Do some setup
cur.executescript('''
DROP TABLE IF EXISTS User;
DROP TABLE IF EXISTS Member;
DROP TABLE IF EXISTS Course;

CREATE TABLE User (
    id     INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name   TEXT UNIQUE
);

CREATE TABLE Course (
    id     INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    title  TEXT UNIQUE
);

CREATE TABLE Member (
    user_id     INTEGER,
    course_id   INTEGER,
    role        INTEGER,
    PRIMARY KEY (user_id, course_id)
)
''')

fname = input('Enter file name: ')
if len(fname) < 1:
    fname = 'roster_data.json'

# [
#   [ "Charley", "si110", 1 ],
#   [ "Mea", "si110", 0 ],

str_data = open(fname).read()
json_data = json.loads(str_data)

for entry in json_data:

    name = entry[0];
    title = entry[1];
    role = entry[2];

    print((name, title, role))

    cur.execute('''INSERT OR IGNORE INTO User (name)
        VALUES ( ? )''', ( name, ) )
    cur.execute('SELECT id FROM User WHERE name = ? ', (name, ))
    user_id = cur.fetchone()[0]

    cur.execute('''INSERT OR IGNORE INTO Course (title)
        VALUES ( ? )''', ( title, ) )
    cur.execute('SELECT id FROM Course WHERE title = ? ', (title, ))
    course_id = cur.fetchone()[0]

    cur.execute('''INSERT OR REPLACE INTO Member
        (user_id, course_id, role) VALUES ( ?, ?, ? )''',
        ( user_id, course_id, role, ) )

    conn.commit()

```

Output from the code above in the console terminal:

```
Run:  Ch15w4Assignment
/Users/alex/PycharmProjects/PY4E/venv/bin/python /Users/alex/PycharmProjects/PY4E/Ch15w4Assignment.py
Enter file name:
('Miao', 'si110', 1)
('Wanessa', 'si110', 0)
('Elora', 'si110', 0)
('Kasper', 'si110', 0)
('Sukhveer', 'si110', 0)
('Muneeba', 'si110', 0)
('Sandy', 'si110', 0)
('Leighton', 'si110', 0)
('Aiden', 'si110', 0)
('Vladislav', 'si110', 0)
('Aamina', 'si110', 0)
('Aadam', 'si110', 0)
('Oluwanifemi', 'si110', 0)
('Modu', 'si110', 0)
('Shwetika', 'si110', 0)
('Leevi', 'si110', 0)
('Jake', 'si110', 0)
('Maddison', 'si110', 0)
('Teos', 'si110', 0)
('Raphael', 'si110', 0)
```

Results from running the Select statement provided in the prompt, in the SQLite browser, the correct answer is the top row:



E.5: Week 5 - “Retrieving GeoData” Assignment: The “geodata.zip” file containing geoload.py, geodump.py, where.data, where.js, and where.html is provided in the “Sample code” folder that was downloaded at the beginning of the specialization. For this assignment you need to add a location (school) to the where.data file, then run the geoload.py file to lookup all of the entries in the where.data (including the new one) and produce the geodata.sqlite database. After, run the geodump.py file to read the database and produce the where.js file. Finally, you can open where.html to visualize the map.

Prompt for the assignment:

Welcome Alex Huerta from Using Databases with Python

Retrieving GEOData

Download the code from http://www.py4e.com/code3/geodata.zip - then unzip the file and edit where.data to add an address nearby where you live - don't reveal where you live. Then run the geoload.py to lookup all of the entries in where.data (including the new one) and produce the geodata.sqlite. Then run geodump.py to read the database and produce where.js. You can run the programs and then scroll back to take your screen shots when the program finishes. Then open where.html to visualize the map. Take screen shots as described below. Make sure that your added location shows in all three of your screen shots.

This is a relatively simple assignment. Don't take off points for little mistakes. If they seem to have done the assignment give them full credit. Feel free to make suggestions if there are small mistakes. Please keep your comments positive and useful. If you do not take grading seriously, the instructors may delete your response and you will lose points.

Please Upload Your Submission:

Screen shot (JPG or PNG - Maximum 1MB) of the geoload.py program output in the command line showing your added location

Choose File No file chosen

(Please use PNG or JPG files)

Screen shot (JPG or PNG - Maximum 1MB) of the geodump.py program output in the command line showing your added location

Choose File No file chosen

(Please use PNG or JPG files)

Screen shot (JPG or PNG - Maximum 1MB) of the map zoomed into the location that you added.) of your home page.

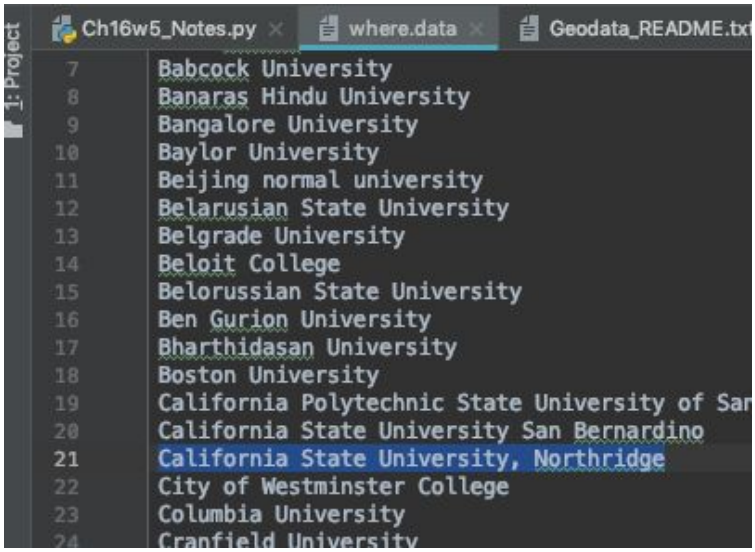
Choose File No file chosen

(Please use PNG or JPG files)

Enter optional comments below

Submit

The location I chose to add (Cal State University, Northridge) to the where.data file:

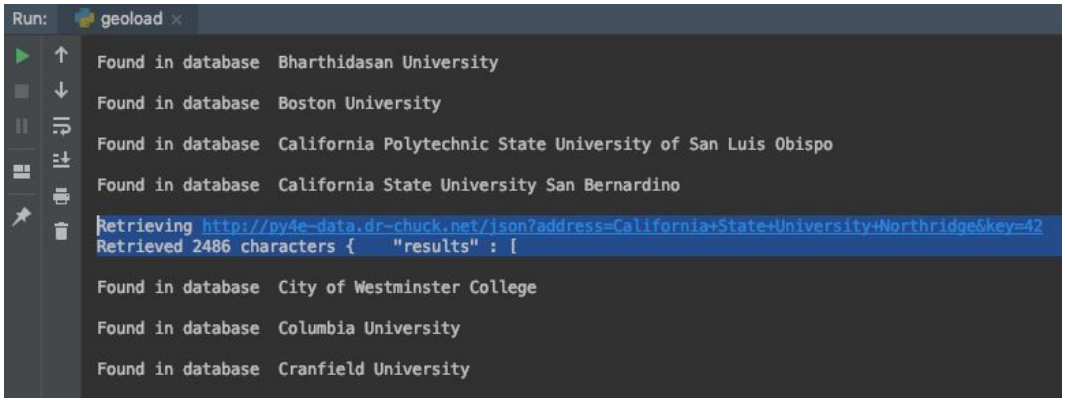


Geoload.py creates the geodata.sqlite database, uses the API endpoint (http.//py4e-data.dr.chuck.net/json?) which has a static subset of Google's GeoCode, then reads the where.data file to lookup all of the entries (including Cal State University Northridge), the program ultimately retrieves the locations from the JSON data:

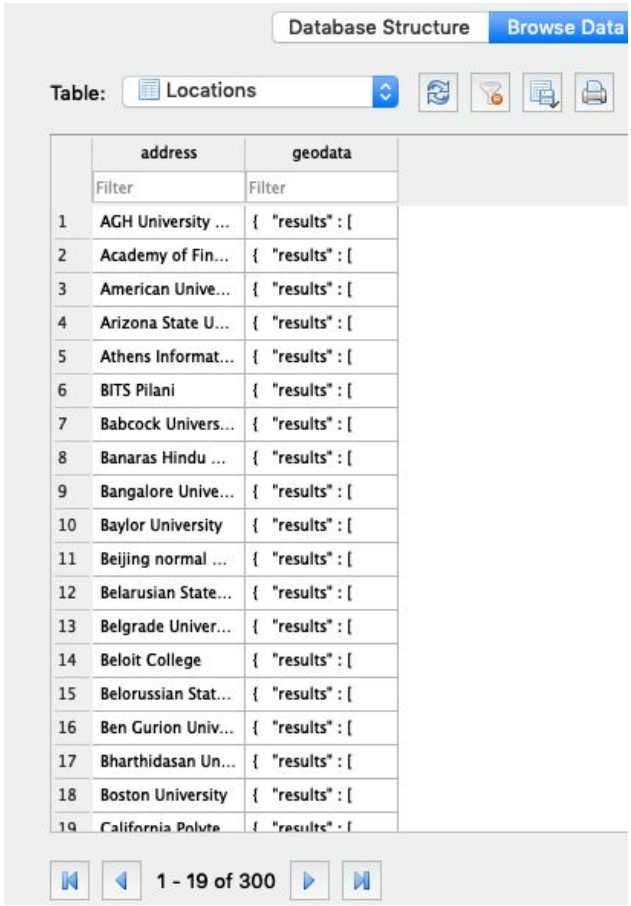
Ch16w5_Notes.py | geoload.py | geodump.py | where.html | where.js | where.data

```
import urllib.request, urllib.parse, urllib.error
import http
import sqlite3
import json
import time
import ssl
import sys

api_key = False
# If you have a Google Places API key, enter it here
# api_key = 'AIzaSy___IDByT70'

if api_key is False:
    api_key = 42
    serviceurl = "http://py4e-data.dr-chuck.net/json?"
else :
    serviceurl = "https://maps.googleapis.com/maps/api/geocode/json?"
# Additional detail for urllib
# http.client.HTTPConnection.debuglevel = 1
conn = sqlite3.connect('geodata.sqlite')
cur = conn.cursor()

cur.execute('''
CREATE TABLE IF NOT EXISTS Locations (address TEXT, geodata TEXT)''')

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

fh = open("where.data")
count = 0
for line in fh:
    if count > 200 :
        print('Retrieved 200 locations, restart to retrieve more')
        break

    address = line.strip()
    print('')
    cur.execute("SELECT geodata FROM Locations WHERE address= ?",
        (memoryview(address.encode()), ))

    try:
        data = cur.fetchone()[0]
        print("Found in database ",address)
        continue
    except:
        pass

    parms = dict()
    parms["address"] = address
    if api_key is not False: parms['key'] = api_key
    url = serviceurl + urllib.parse.urlencode(parms)

    print('Retrieving', url)
    uh = urllib.request.urlopen(url, context=ctx)
    data = uh.read().decode()
    print('Retrieved', len(data), 'characters', data[:20].replace('\n', ' '))
    count = count + 1

    try:
        js = json.loads(data)
    except:
        print(data)  # We print in case unicode causes an error
        continue

    if 'status' not in js or (js['status'] != 'OK' and js['status'] != 'ZERO_RESULTS') :
        print('==== Failure To Retrieve ====')
        print(data)
        break

    cur.execute('''INSERT INTO Locations (address, geodata)
            VALUES ( ?, ? )''', (memoryview(address.encode()), memoryview(data.encode()) ) )
    conn.commit()
    if count % 10 == 0 :
        print('Pausing for a bit...')
        time.sleep(5)

print("Run geodump.py to read the data from the database so you can vizualize it on a map.")
```

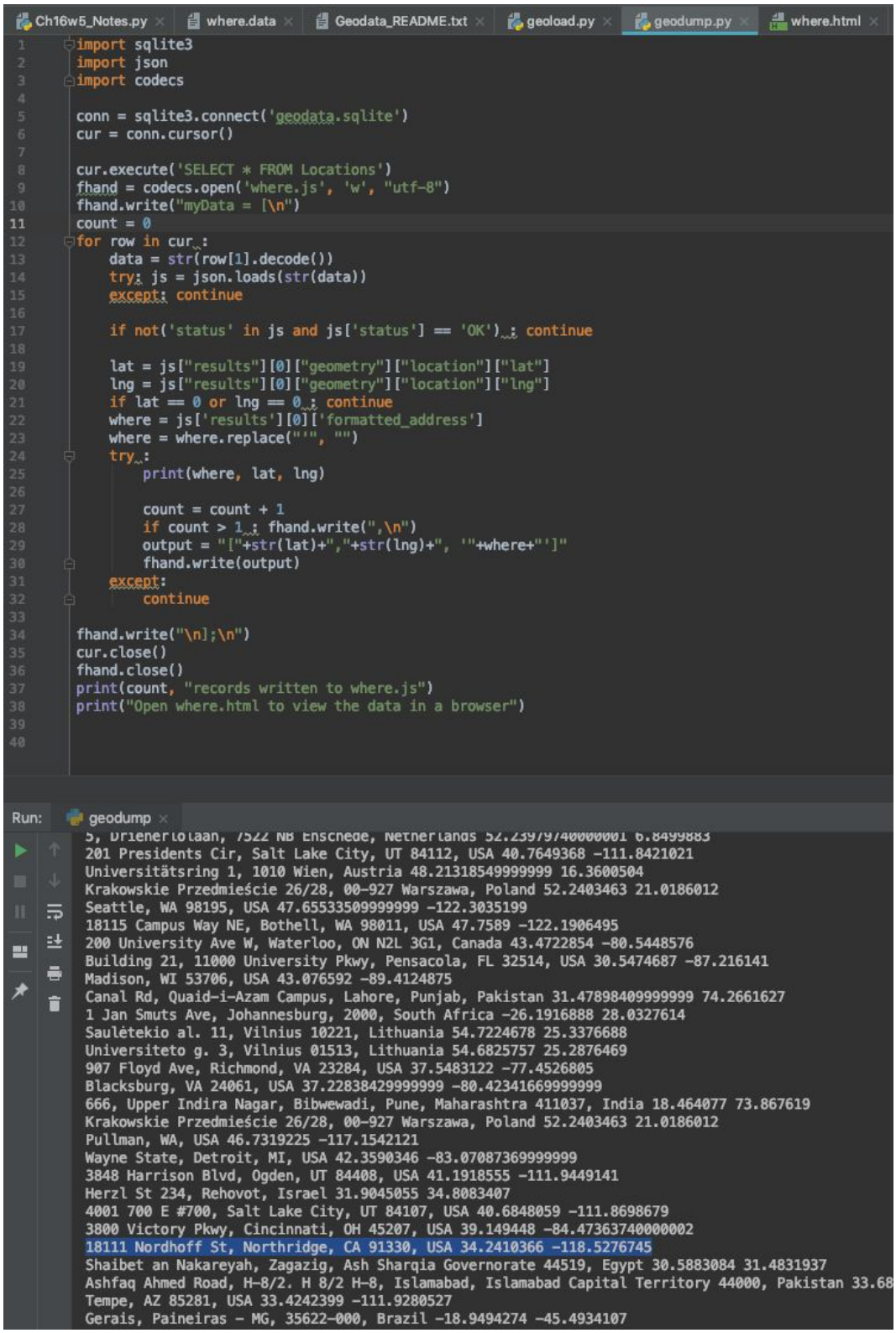
Output from geoload.py showing my added location (screenshot required for submission):



Geodata.sqlite database created by geoload.py:



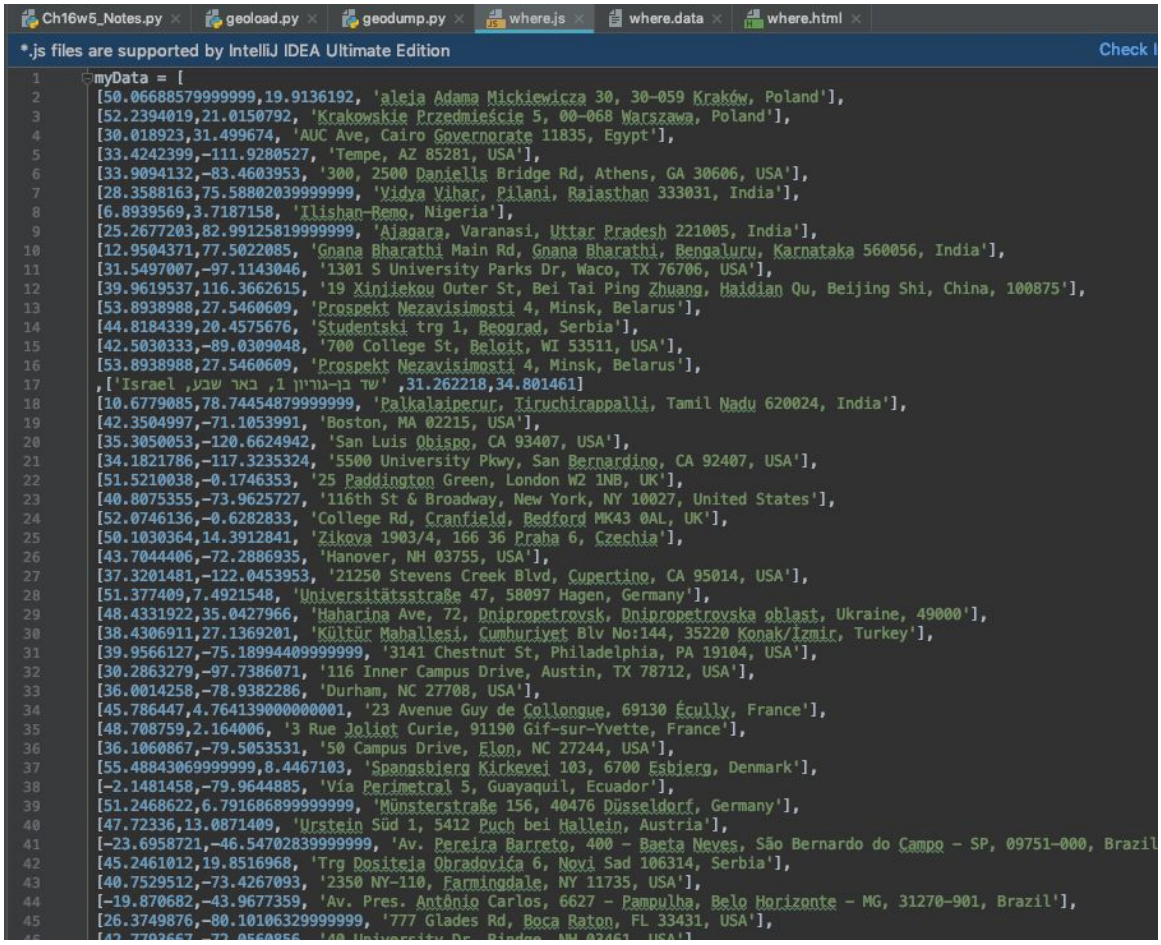
Geodump.py reads the database and produces the where.js file. Geodump.py uses SQL to Select the locations from the database and JSON to retrieve the locations and parse them with their coordinates (screenshot required for submission) :



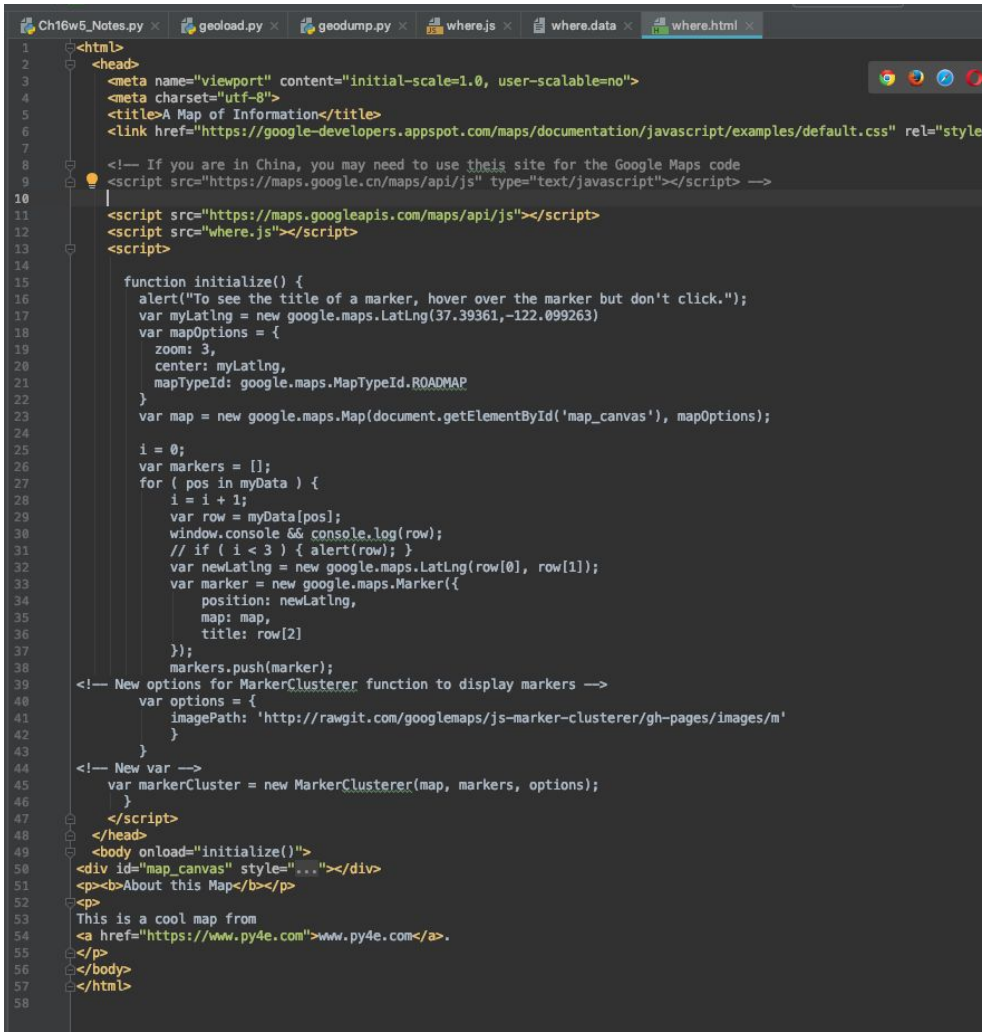
```
import sqlite3
import json
import codecs

conn = sqlite3.connect('geodata.sqlite')
cur = conn.cursor()

cur.execute('SELECT * FROM Locations')
fhand = codecs.open('where.js', 'w', "utf-8")
fhand.write("myData = [\n")
count = 0
for row in cur :
    data = str(row[1].decode())
    try: js = json.loads(str(data))
    except: continue

    if not('status' in js and js['status'] == 'OK') : continue

    lat = js["results"][0]["geometry"]["location"]["lat"]
    lng = js["results"][0]["geometry"]["location"]["lng"]
    if lat == 0 or lng == 0 : continue
    where = js['results'][0]['formatted_address']
    where = where.replace("'", "")
    try :
        print(where, lat, lng)

        count = count + 1
        if count > 1 : fhand.write(",\n")
        output = "["+str(lat)+","+str(lng)+", '"+where+"']"
        fhand.write(output)
    except:
        continue

fhand.write("\n];\n")
cur.close()
fhand.close()
print(count, "records written to where.js")
print("Open where.html to view the data in a browser")
```

```
Run: geodump
5, Drienerlolaan, 7522 NB Enschede, Netherlands 52.2397974000000 6.8499883
201 Presidents Cir, Salt Lake City, UT 84112, USA 40.7649368 -111.8421021
Universitätsring 1, 1010 Wien, Austria 48.21318549999999 16.3600504
Krakowskie Przedmieście 26/28, 00-927 Warszawa, Poland 52.2403463 21.0186012
Seattle, WA 98195, USA 47.65533509999999 -122.3035199
18115 Campus Way NE, Bothell, WA 98011, USA 47.7589 -122.1906495
200 University Ave W, Waterloo, ON N2L 3G1, Canada 43.4722854 -80.5448576
Building 21, 11000 University Pkwy, Pensacola, FL 32514, USA 30.5474687 -87.216141
Madison, WI 53706, USA 43.076592 -89.4124875
Canal Rd, Quaid-i-Azam Campus, Lahore, Punjab, Pakistan 31.47898409999999 74.2661627
1 Jan Smuts Ave, Johannesburg, 2000, South Africa -26.1916888 28.0327614
Saulėtekio al. 11, Vilnius 10221, Lithuania 54.7224678 25.3376688
Universiteto g. 3, Vilnius 01513, Lithuania 54.6825757 25.2876469
907 Floyd Ave, Richmond, VA 23284, USA 37.5483122 -77.4526805
Blacksburg, VA 24061, USA 37.22838429999999 -80.42341669999999
666, Upper Indira Nagar, Bibwewadi, Pune, Maharashtra 411037, India 18.464077 73.867619
Krakowskie Przedmieście 26/28, 00-927 Warszawa, Poland 52.2403463 21.0186012
Pullman, WA, USA 46.7319225 -117.1542121
Wayne State, Detroit, MI, USA 42.3590346 -83.07087369999999
3848 Harrison Blvd, Ogden, UT 84408, USA 41.1918555 -111.9449141
Herzl St 234, Rehovot, Israel 31.9045055 34.8083407
4001 700 E #700, Salt Lake City, UT 84107, USA 40.6848059 -111.8698679
3800 Victory Pkwy, Cincinnati, OH 45207, USA 39.149448 -84.47363740000002
18111 Nordhoff St, Northridge, CA 91330, USA 34.2410366 -118.5276745
Shaibet an Nakareyah, Zagazig, Ash Sharqia Governorate 44519, Egypt 30.5883084 31.4831937
Ashfaq Ahmed Road, H-8/2. H 8/2 H-8, Islamabad, Islamabad Capital Territory 44000, Pakistan 33.68
Tempe, AZ 85281, USA 33.4242399 -111.9280527
Gerais, Paineiras - MG, 35622-000, Brazil -18.9494274 -45.4934107
```

Where.js file, the result from running geodump.py:



Where.html file used to open a browser to link you to the visualization of the location data:




```
<html>
  <head>
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no">
    <meta charset="utf-8">
    <title>A Map of Information</title>
    <link href="https://google-developers.appspot.com/maps/documentation/javascript/examples/default.css" rel="style

    <!-- If you are in China, you may need to use theis site for the Google Maps code
    <script src="https://maps.google.cn/maps/api/js" type="text/javascript"></script> -->

    <script src="https://maps.googleapis.com/maps/api/js"></script>
    <script src="where.js"></script>
    <script>

      function initialize() {
        alert("To see the title of a marker, hover over the marker but don't click.");
        var myLatlng = new google.maps.LatLng(37.39361,-122.099263)
        var mapOptions = {
          zoom: 3,
          center: myLatlng,
          mapTypeId: google.maps.MapTypeId.ROADMAP
        }
        var map = new google.maps.Map(document.getElementById('map_canvas'), mapOptions);

        i = 0;
        var markers = [];
        for ( pos in myData ) {
            i = i + 1;
            var row = myData[pos];
            window.console && console.log(row);
            // if ( i < 3 ) { alert(row); }
            var newLatlng = new google.maps.LatLng(row[0], row[1]);
            var marker = new google.maps.Marker({
                position: newLatlng,
                map: map,
                title: row[2]
            });
            markers.push(marker);
<!-- New options for MarkerClusterer function to display markers -->
        var options = {
            imagePath: 'http://rawgit.com/googlemaps/js-marker-clusterer/gh-pages/images/m'
            }
        }
<!-- New var -->
    var markerCluster = new MarkerClusterer(map, markers, options);
      }
    </script>
  </head>
  <body onload="initialize()">
<div id="map_canvas" style="..."></div>
<p><b>About this Map</b></p>
<p>
This is a cool map from
<a href="https://www.py4e.com">www.py4e.com</a>.
</p>
</body>
</html>
```

Link that your are lead to after running the where.html file (in chrome), shows the added location’s address (retrieved using the API), visualization of data (screenshot required for submission):

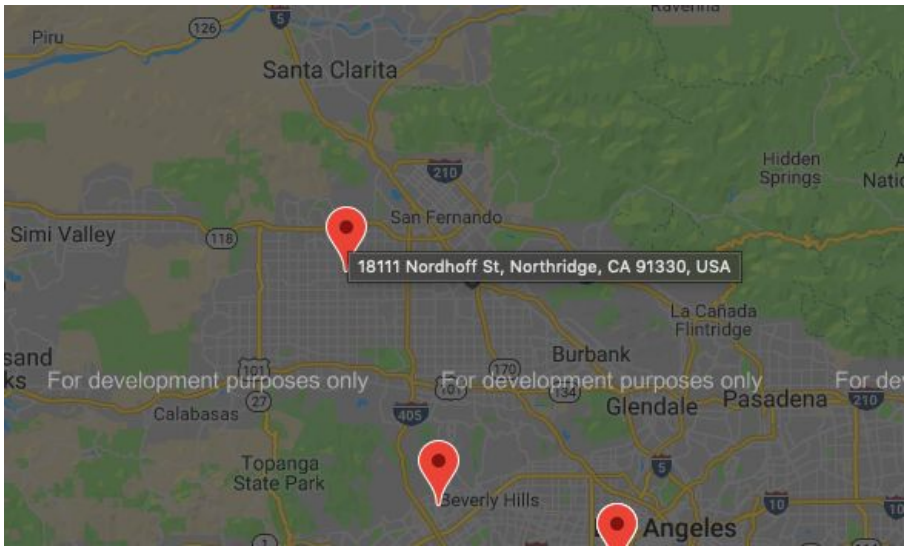
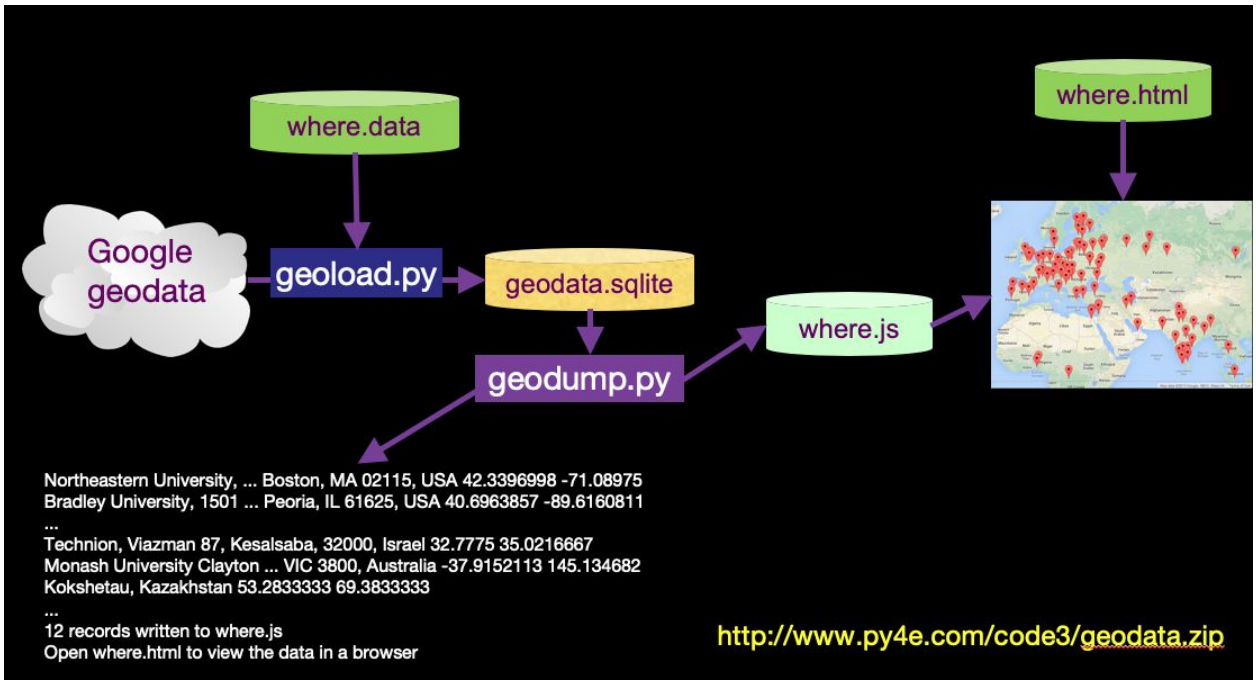


Diagram from the Ch.16 lecture slides, shows how each of the pages of code provided relate to each other, helps to understand what they do:



F. Course 5:

F.1: Week 2 - Page Rank Assignment:

Prompt for the assignment:

Welcome Alex Huerta from Capstone: Retrieving, Processing, and Visualizing Data with Python

Page Rank

First you will spider 100 pages from http://python-data.dr-chuck.net/ run the page rank algorithm and take some screen shots. Then you will reset the spider process and spider 100 pages from any other site on the Internet, run the page rank algorithm, and take some screen shots.

Don't take off points for little mistakes. If they seem to have done the assignment give them full credit. Feel free to make suggestions if there are small mistakes. Please keep your comments positive and useful. The course staff will assign the last 30% of the grade and it may take a few days to get graded.

Please Upload Your Submission:

Screenshot of spdump.py crawling 100 pages of python-data.dr-chuck.net including rows retrieved. You must run sprank.py first in order to show the page rankings.

Choose File No file chosen

(Please use PNG or JPG files)

Screenshot showing force.html restricted to 25 nodes for python-data.dr-chuck.net. You must show the starting url.

Choose File No file chosen

(Please use PNG or JPG files)

Screenshot of spdump.py crawling 100 pages of a website of your choosing (other than dr. Chuck's website) including rows retrieved. You must run sprank.py first in order to show the page rankings.

Choose File No file chosen

(Please use PNG or JPG files)

Screenshot of force.html restricted to 25 nodes for your chosen website. You must show the starting url.

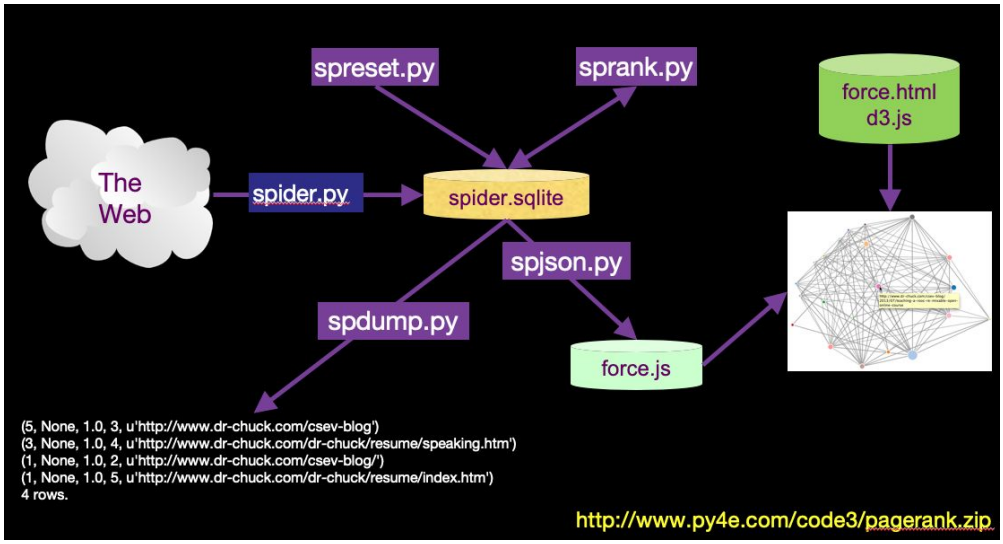
Choose File No file chosen

(Please use PNG or JPG files)

Enter optional comments below

Submit

Diagram from the Ch.16 lecture slides, shows how each of the pages of code provided relate to each other, helps to understand what they do:



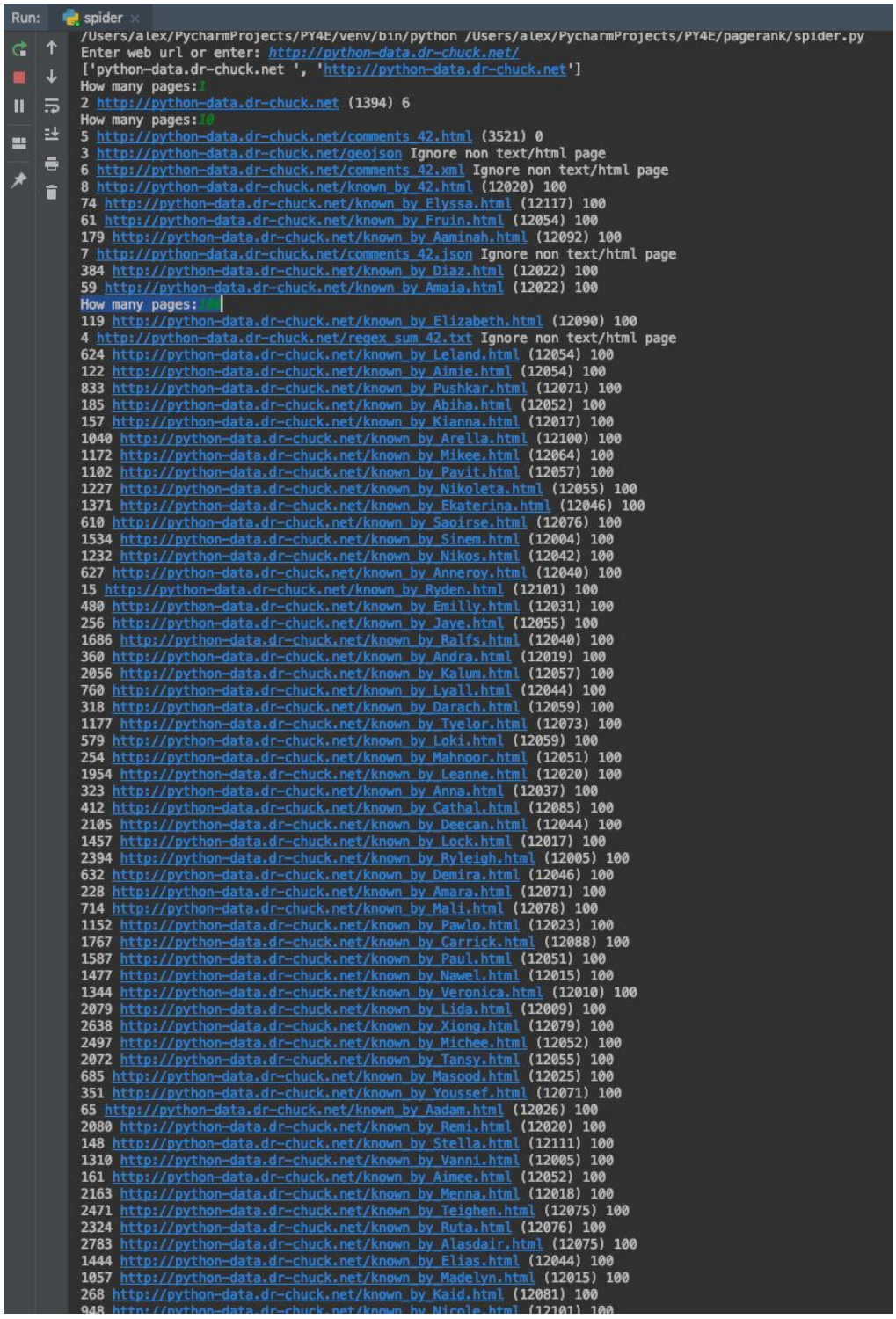
Spider.py (pt. 1):


```
import sqlite3
import urllib.error
import ssl
from urllib.parse import urljoin
from urllib.parse import urlparse
from urllib.request import urlopen
from bs4 import BeautifulSoup

# Building databases, reading urls, parsing html w/beautiful soup
# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

conn = sqlite3.connect('spider.sqlite')
cur = conn.cursor()

cur.execute('''CREATE TABLE IF NOT EXISTS Pages
    (id INTEGER PRIMARY KEY, url TEXT UNIQUE, html TEXT,
     error INTEGER, old_rank REAL, new_rank REAL)''')

cur.execute('''CREATE TABLE IF NOT EXISTS Links
    (from_id INTEGER, to_id INTEGER)''')

cur.execute('''CREATE TABLE IF NOT EXISTS Webs (url TEXT UNIQUE)''')

# Check to see if we are already in progress...
cur.execute('SELECT id,url FROM Pages WHERE html is NULL and error is NULL ORDER BY RANDOM() LIMIT 1')
row = cur.fetchone()
if row is not None:
    print("Restarting existing crawl.  Remove spider.sqlite to start a fresh crawl.")
else :
    starturl = input('Enter web url or enter: ')
    if ( len(starturl) < 1 ) : starturl = 'http://www.dr-chuck.com/'
    if ( starturl.endswith('/') ) : starturl = starturl[:-1]
    web = starturl
    if ( starturl.endswith('.htm') or starturl.endswith('.html') ) :
        pos = starturl.rfind('/')
        web = starturl[:pos]

    if ( len(web) > 1 ) :
        cur.execute('INSERT OR IGNORE INTO Webs (url) VALUES ( ? )', ( web, ) )
        cur.execute('INSERT OR IGNORE INTO Pages (url, html, new_rank) VALUES ( ?, NULL, 1.0 )', ( starturl, ) )
        conn.commit()

# Get the current webs
cur.execute('''SELECT url FROM Webs''')
webs = list()
for row in cur:
    webs.append(str(row[0]))

print(webs)

many = 0
while True:
    if ( many < 1 ) :
        sval = input('How many pages:')
        if ( len(sval) < 1 ) : break
        many = int(sval)
    many = many - 1

    cur.execute('SELECT id,url FROM Pages WHERE html is NULL and error is NULL ORDER BY RANDOM() LIMIT 1')
    try:
        row = cur.fetchone()
        # print row
        fromid = row[0]
        url = row[1]
    except:
        print('No unretrieved HTML pages found')
        many = 0
        break

    print(fromid, url, end=' ')

    # If we are retrieving this page, there should be no links from it
    cur.execute('DELETE from Links WHERE from_id=?', (fromid, ) )
    try:
        document = urlopen(url, context=ctx)
```

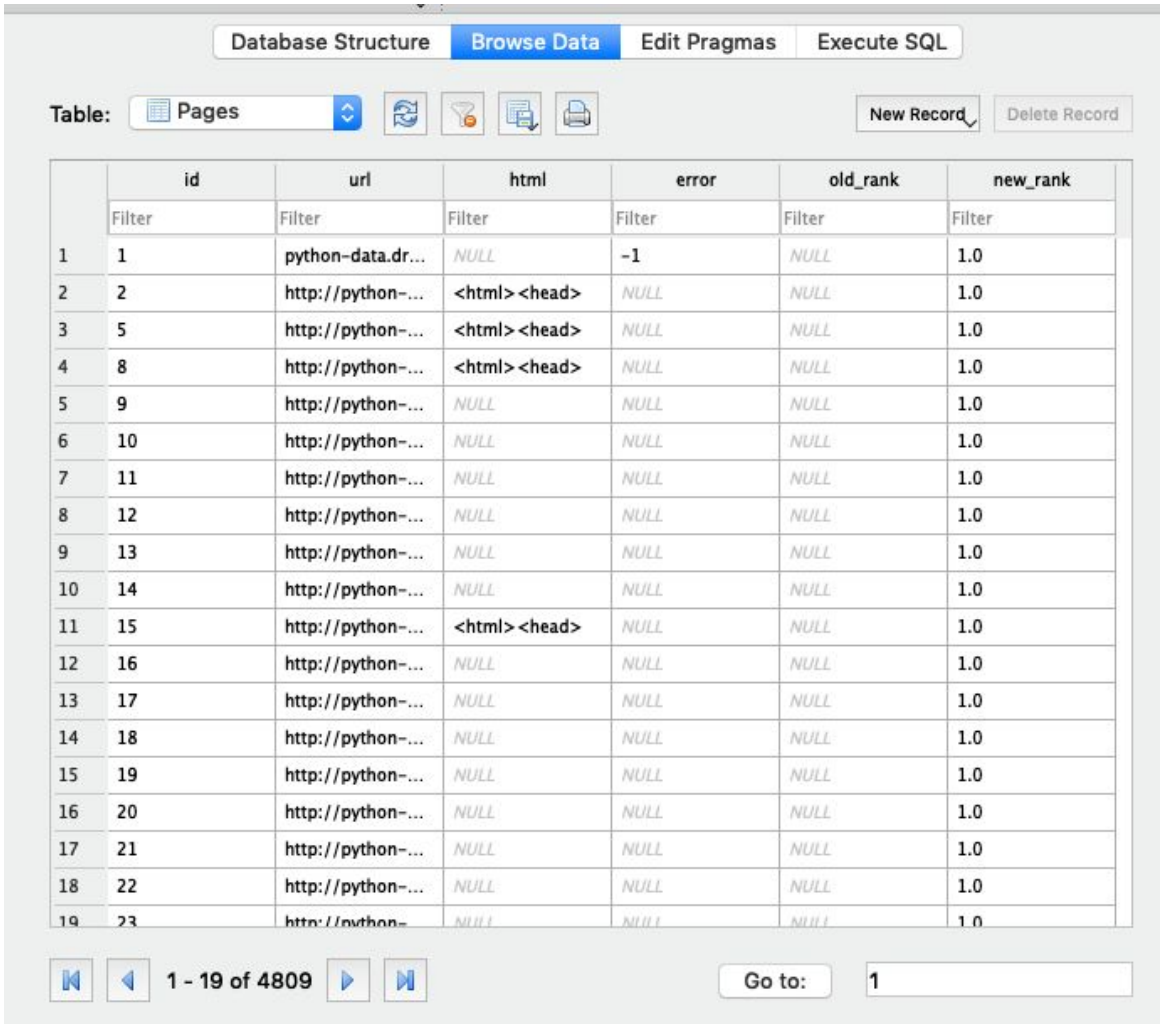
Spider.py (pt.2):


```
    try:
        document = urlopen(url, context=ctx)

        html = document.read()
        if document.getcode() != 200 :
            print("Error on page: ",document.getcode())
            cur.execute('UPDATE Pages SET error=? WHERE url=?', (document.getcode(), url) )

        if 'text/html' != document.info().get_content_type() :
            print("Ignore non text/html page")
            cur.execute('DELETE FROM Pages WHERE url=?', ( url, ) )
            cur.execute('UPDATE Pages SET error=0 WHERE url=?', (url, ) )
            conn.commit()
            continue

        print('('+str(len(html))+')', end=' ')

        soup = BeautifulSoup(html, "html.parser")
    except KeyboardInterrupt:
        print('')
        print('Program interrupted by user...')
        break
    except:
        print("Unable to retrieve or parse page")
        cur.execute('UPDATE Pages SET error=-1 WHERE url=?', (url, ) )
        conn.commit()
        continue

    cur.execute('INSERT OR IGNORE INTO Pages (url, html, new_rank) VALUES ( ?, NULL, 1.0 )', ( url, ) )
    cur.execute('UPDATE Pages SET html=? WHERE url=?', (memoryview(html), url ) )
    conn.commit()

    # Retrieve all of the anchor tags
    tags = soup('a')
    count = 0
    for tag in tags:
        href = tag.get('href', None)
        if ( href is None ) : continue
        # Resolve relative references like href="/contact"
        up = urlparse(href)
        if ( len(up.scheme) < 1 ) :
            href = urljoin(url, href)
        ipos = href.find('#')
        if ( ipos > 1 ) : href = href[:ipos]
        if ( href.endswith('.png') or href.endswith('.jpg') or href.endswith('.gif') ) : continue
        if ( href.endswith('/') ) : href = href[:-1]
        # print href
        if ( len(href) < 1 ) : continue

        # Check if the URL is in any of the webs
        found = False
        for web in webs:
            if ( href.startswith(web) ) :
                found = True
                break
        if not found : continue

        cur.execute('INSERT OR IGNORE INTO Pages (url, html, new_rank) VALUES ( ?, NULL, 1.0 )', ( href, ) )
        count = count + 1
        conn.commit()

        cur.execute('SELECT id FROM Pages WHERE url=? LIMIT 1', ( href, ))
        try:
            row = cur.fetchone()
            toid = row[0]
        except:
            print('Could not retrieve id')
            continue
        # print fromid, toid
        cur.execute('INSERT OR IGNORE INTO Links (from_id, to_id) VALUES ( ?, ? )', ( fromid, toid ) )


    print(count)

cur.close()
```

Results from running spider.py, “spidering” 100 pages from http://python-data.dr-chuck.net:



Spider.sqlite database created by running spider.py:



	id	url	html	error	old_rank	new_rank
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	python-data.dr...	NULL	-1	NULL	1.0
2	2	http://python-...	<html><head>	NULL	NULL	1.0
3	5	http://python-...	<html><head>	NULL	NULL	1.0
4	8	http://python-...	<html><head>	NULL	NULL	1.0
5	9	http://python-...	NULL	NULL	NULL	1.0
6	10	http://python-...	NULL	NULL	NULL	1.0
7	11	http://python-...	NULL	NULL	NULL	1.0
8	12	http://python-...	NULL	NULL	NULL	1.0
9	13	http://python-...	NULL	NULL	NULL	1.0
10	14	http://python-...	NULL	NULL	NULL	1.0
11	15	http://python-...	<html><head>	NULL	NULL	1.0
12	16	http://python-...	NULL	NULL	NULL	1.0
13	17	http://python-...	NULL	NULL	NULL	1.0
14	18	http://python-...	NULL	NULL	NULL	1.0
15	19	http://python-...	NULL	NULL	NULL	1.0
16	20	http://python-...	NULL	NULL	NULL	1.0
17	21	http://python-...	NULL	NULL	NULL	1.0
18	22	http://python-...	NULL	NULL	NULL	1.0
19	23	http://python-	NULL	NULL	NULL	1.0

Sprank.py (pt.1):

Course5w2_Notes.py | spider.py | sprank.py | spdump.py | spjson.py | spider.js

```
import sqlite3
# page rank algorithm

conn = sqlite3.connect('spider.sqlite')
cur = conn.cursor()

# Find the ids that send out page rank - we only are interested
# in pages in the SCC that have in and out links
cur.execute('''SELECT DISTINCT from_id FROM Links''')
from_ids = list()
for row in cur:
    from_ids.append(row[0])

# Find the ids that receive page rank
to_ids = list()
links = list()
cur.execute('''SELECT DISTINCT from_id, to_id FROM Links''')
for row in cur:
    from_id = row[0]
    to_id = row[1]
    if from_id == to_id : continue
    if from_id not in from_ids : continue
    if to_id not in from_ids : continue
    links.append(row)
    if to_id not in to_ids : to_ids.append(to_id)

# Get latest page ranks for strongly connected component
prev_ranks = dict()
for node in from_ids:
    cur.execute('''SELECT new_rank FROM Pages WHERE id = ?''', (node, ))
    row = cur.fetchone()
    prev_ranks[node] = row[0]

sval = input('How many iterations:')
many = 1
if ( len(sval) > 0 ) : many = int(sval)

# Sanity check
if len(prev_ranks) < 1 :
    print("Nothing to page rank.  Check data.")
    quit()

# Lets do Page Rank in memory so it is really fast
for i in range(many):
    # print prev_ranks.items()[:5]
    next_ranks = dict();
    total = 0.0
    for (node, old_rank) in list(prev_ranks.items()):
        total = total + old_rank
        next_ranks[node] = 0.0
    # print total

    # Find the number of outbound links and sent the page rank down each
    for (node, old_rank) in list(prev_ranks.items()):
        # print node, old_rank
        give_ids = list()
        for (from_id, to_id) in links:
            if from_id != node : continue
            #  print '   ',from_id,to_id

            if to_id not in to_ids: continue
            give_ids.append(to_id)
        if ( len(give_ids) < 1 ) : continue
        amount = old_rank / len(give_ids)
        # print node, old_rank,amount, give_ids

        for id in give_ids:
            next_ranks[id] = next_ranks[id] + amount

    newtot = 0
    for (node, next_rank) in list(next_ranks.items()):
        newtot = newtot + next_rank
    evap = (total - newtot) / len(next_ranks)
```

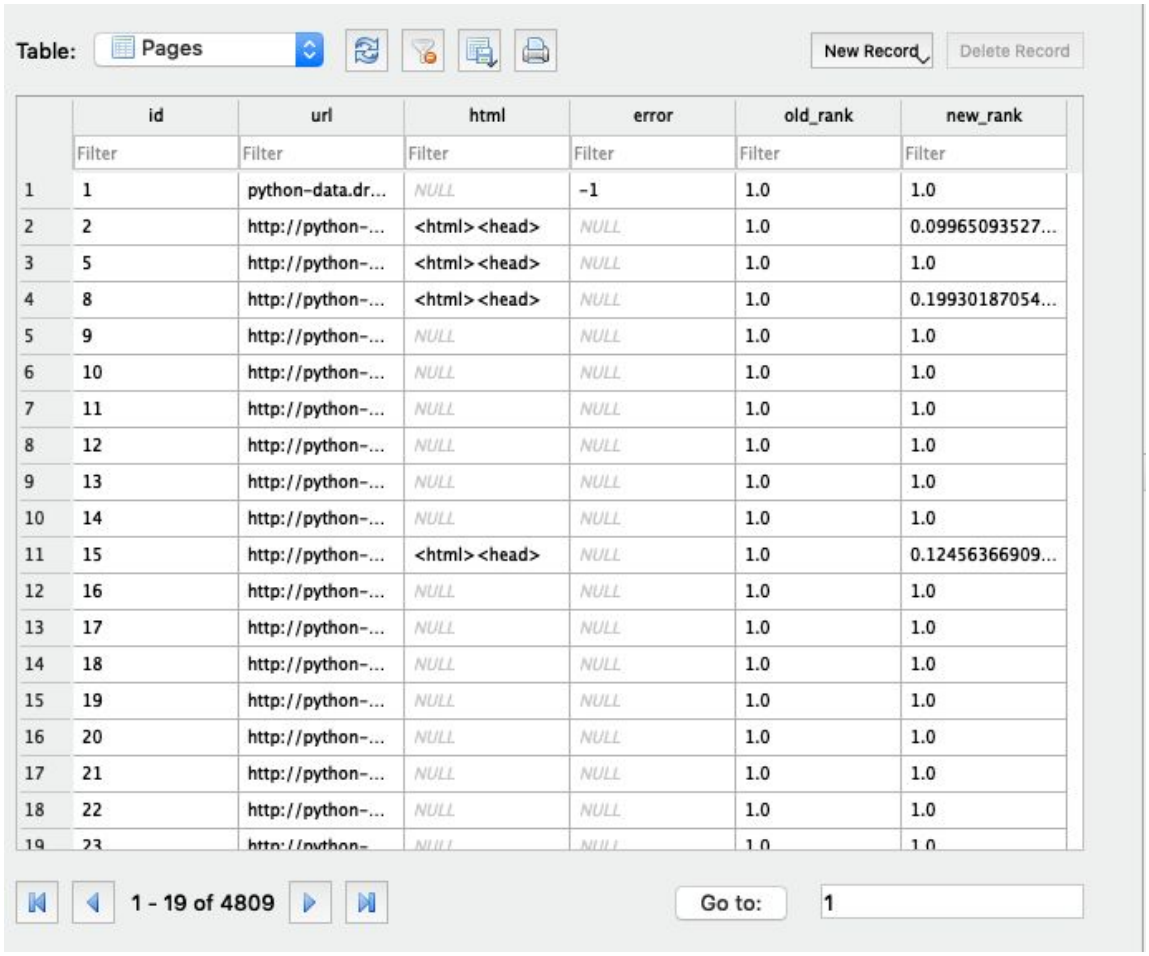
Sprank.py (pt.2):

```
74
75      # print newtot, evap
76      for node in next_ranks:
77          next_ranks[node] = next_ranks[node] + evap
78
79      newtot = 0
80      for (node, next_rank) in list(next_ranks.items()):
81          newtot = newtot + next_rank
82
83      # Compute the per-page average change from old rank to new rank
84      # As indication of convergence of the algorithm
85      totdiff = 0
86      for (node, old_rank) in list(prev_ranks.items()):
87          new_rank = next_ranks[node]
88          diff = abs(old_rank-new_rank)
89          totdiff = totdiff + diff
90
91      avediff = totdiff / len(prev_ranks)
92      print(i+1, avediff)
93
94      # rotate
95      prev_ranks = next_ranks
96
97  # Put the final ranks back into the database
98  print(list(next_ranks.items())[:5])
99  cur.execute('''UPDATE Pages SET old_rank=new_rank''')
100 for (id, new_rank) in list(next_ranks.items()):
101     cur.execute('''UPDATE Pages SET new_rank=? WHERE id=?''', (new_rank, id))
102 conn.commit()
103 cur.close()
```

Results from running sprank.py (100 iterations to normalize the ranks as best as possible):

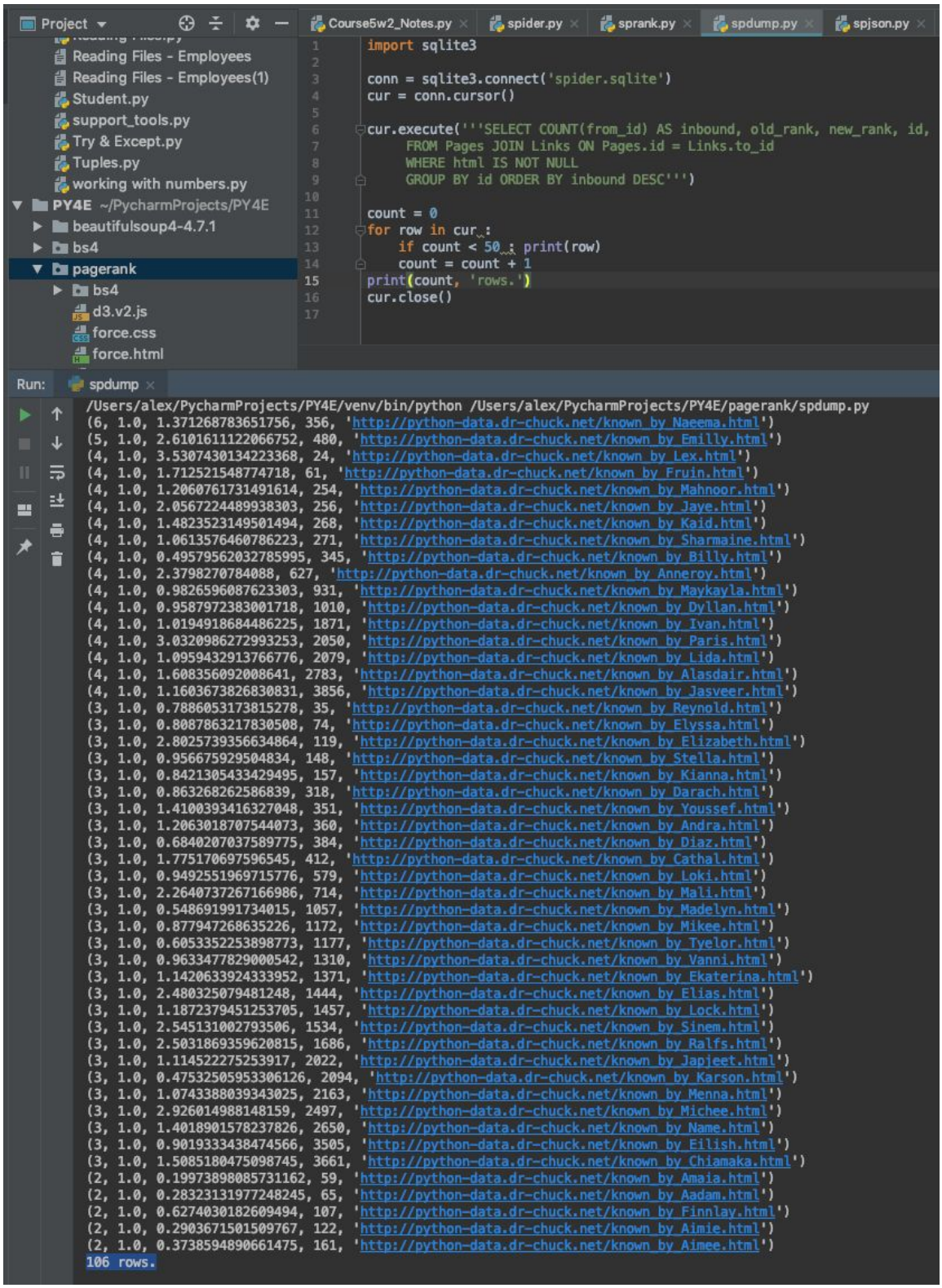
```
Run:  sprank
/Users/alex/PycharmProjects/PY4E/venv/bin/python /Users/alex/PycharmProjects/PY4E/pagerank/sprank.py
How many iterations:100
1 0.4426507484446251
2 0.2592619011216506
3 0.15869898093894685
4 0.10951604950291145
5 0.07608047783909237
6 0.052714671803645974
7 0.04167050536391932
8 0.030501604299477708
9 0.02137668589156627
10 0.014286856560368844
11 0.009521795303278265
12 0.0066312847475213195
13 0.004627829426680675
14 0.003568134956289162
15 0.0026476388603147182
16 0.001852592337867123
17 0.0012193664985869254
18 0.0008821265283382987
19 0.0006666287245819747
20 0.0004713811116997684
21 0.0003209151812537309 5
22 0.0002371333704713689
23 0.0001551972558513852 8
24 0.0001088305621033465 9
25 8.13962367131095e-05
26 6.6379995323734e-05
27 5.065940166152224e-05
```

Spider.sqlite database, after running sprank.py, notice the rank values:

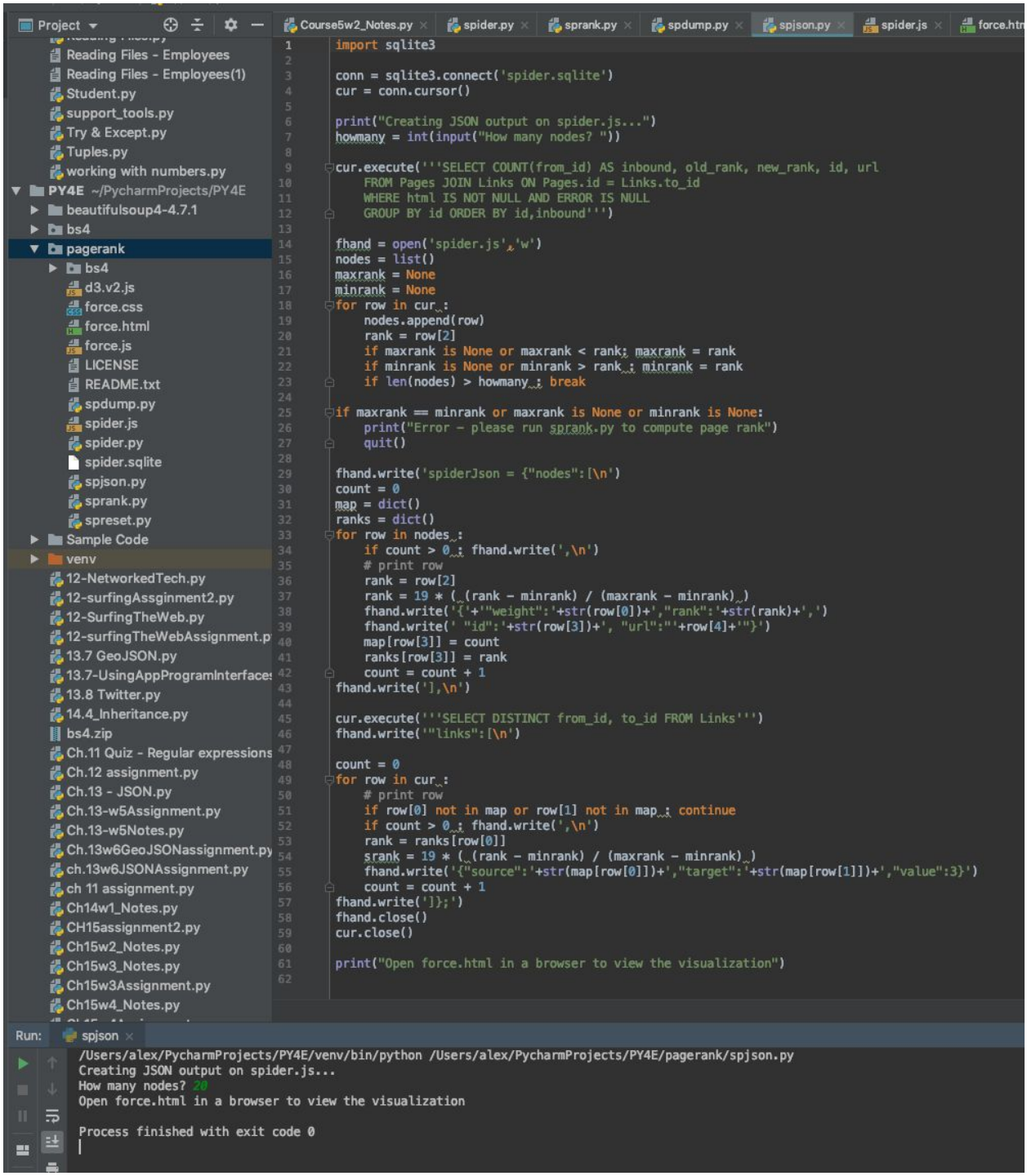


	id	url	html	error	old_rank	new_rank
1	1	python-data.dr...	NULL	-1	1.0	1.0
2	2	http://python-...	<html><head>	NULL	1.0	0.09965093527...
3	5	http://python-...	<html><head>	NULL	1.0	1.0
4	8	http://python-...	<html><head>	NULL	1.0	0.19930187054...
5	9	http://python-...	NULL	NULL	1.0	1.0
6	10	http://python-...	NULL	NULL	1.0	1.0
7	11	http://python-...	NULL	NULL	1.0	1.0
8	12	http://python-...	NULL	NULL	1.0	1.0
9	13	http://python-...	NULL	NULL	1.0	1.0
10	14	http://python-...	NULL	NULL	1.0	1.0
11	15	http://python-...	<html><head>	NULL	1.0	0.12456366909...
12	16	http://python-...	NULL	NULL	1.0	1.0
13	17	http://python-...	NULL	NULL	1.0	1.0
14	18	http://python-...	NULL	NULL	1.0	1.0
15	19	http://python-...	NULL	NULL	1.0	1.0
16	20	http://python-...	NULL	NULL	1.0	1.0
17	21	http://python-...	NULL	NULL	1.0	1.0
18	22	http://python-...	NULL	NULL	1.0	1.0
19	23	http://python-	NULL	NULL	1.0	1.0

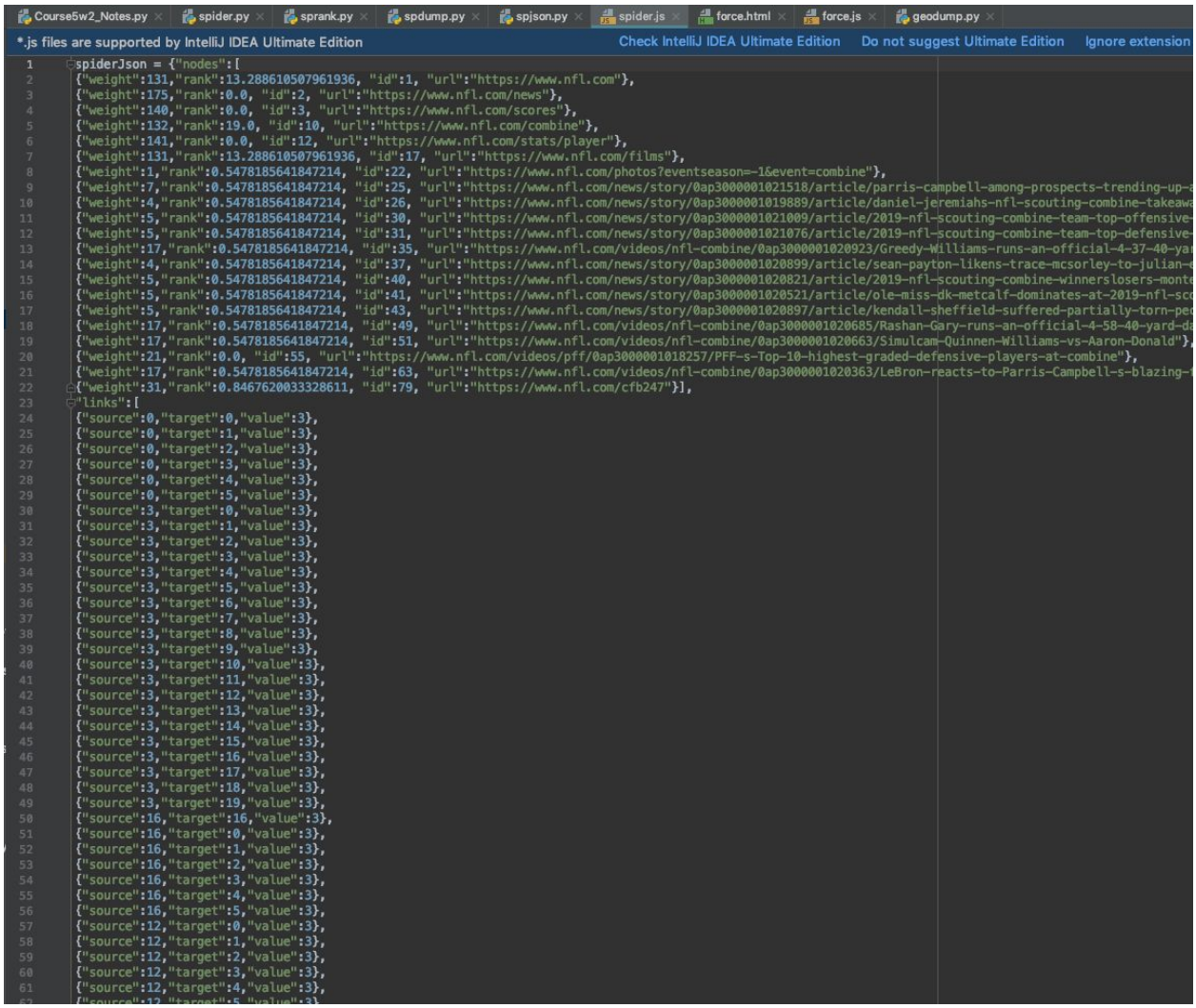
Spdump.py and output in terminal (Select statement w/JOIN to retrieve web page counts (frequencies), old rank, new rank, web page ID, URL of the web page, then prints it to the screen), screenshot needed for assignment:



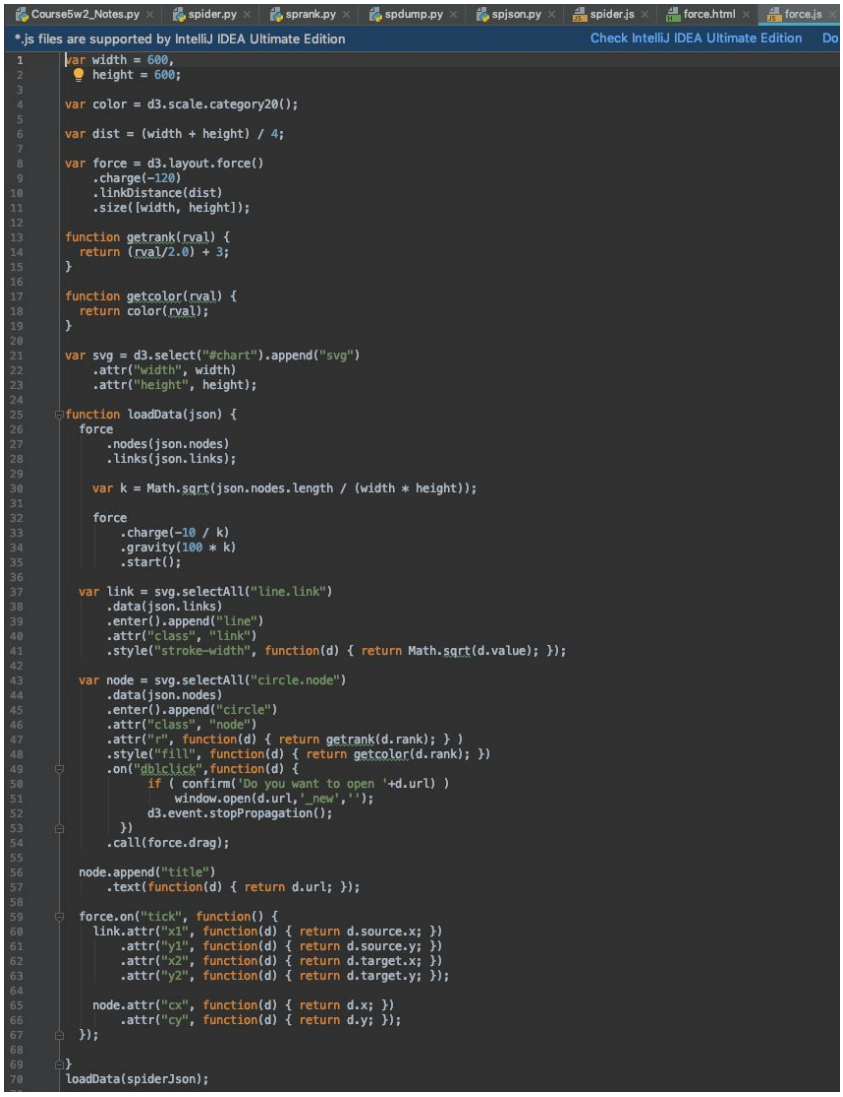
Spjson.py, asking for 20 nodes:



Spider.js, created from running spjson.py, information on the 20 nodes, feeds data to force.js and force.html (used screenshot from nfl data instead of Dr. Chucks website because I didn't want to run the whole over again on my network, they look very similar, just wanted to show you how the data looks)



Force.js (d3 library javascript code, provided by instructor), contains design specifications for visualizations, feeds force.html:



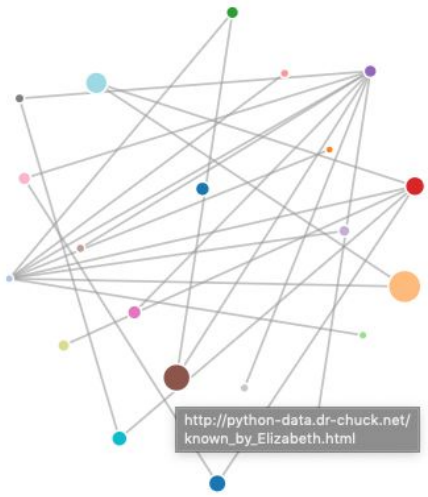

```
var width = 600,
    height = 600;

var color = d3.scale.category20();

var dist = (width + height) / 4;

var force = d3.layout.force()
    .charge(-120)
    .linkDistance(dist)
    .size([width, height]);

function getrank(rval) {
  return (rval/2.0) + 3;
}

function getcolor(rval) {
  return color(rval);
}

var svg = d3.select("#chart").append("svg")
    .attr("width", width)
    .attr("height", height);

function loadData(json) {
  force
      .nodes(json.nodes)
      .links(json.links);

    var k = Math.sqrt(json.nodes.length / (width * height));

    force
        .charge(-10 / k)
        .gravity(100 * k)
        .start();

  var link = svg.selectAll("line.link")
      .data(json.links)
      .enter().append("line")
      .attr("class", "link")
      .style("stroke-width", function(d) { return Math.sqrt(d.value); });

  var node = svg.selectAll("circle.node")
      .data(json.nodes)
      .enter().append("circle")
      .attr("class", "node")
      .attr("r", function(d) { return getrank(d.rank); } )
      .style("fill", function(d) { return getcolor(d.rank); })
      .on("dblclick",function(d) {
            if ( confirm('Do you want to open '+d.url) )
                window.open(d.url,'_new','');
            d3.event.stopPropagation();
        })
      .call(force.drag);

  node.append("title")
      .text(function(d) { return d.url; });

 force.on("tick", function() {
    link.attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });

    node.attr("cx", function(d) { return d.x; })
        .attr("cy", function(d) { return d.y; });
  });

}
loadData(spiderJson);
```

Force.html, notice the src's (sources), browser links in right corner:


```
<!DOCTYPE html>
<html>
  <head>
    <title>Force-Directed Layout</title>
    <script type="text/javascript" src="d3.v2.js"></script>
    <script type="text/javascript" src="spider.js"></script>
    <link type="text/css" rel="stylesheet" href="force.css"/>
  </head>
  <body style="...">
    <script>
        document.write("<p>Starting url: "+spiderJson.nodes[0].url+"</p>");
    </script>
    <div id="chart" style="..."></div>
    <script type="text/javascript" src="force.js"></script>
    <p>If you don't see a chart above, check the JavaScript console. You may
    need to use a different browser.</p>
  </body>
</html>
```

Visualization for Dr. Chuck's website, result from selecting the Chrome browser in force.html (screenshot need for assignment):

Starting url: http://python-data.dr-chuck.net/comments_42.html



If you don't see a chart above, check the JavaScript console. You may need to use a different browser.

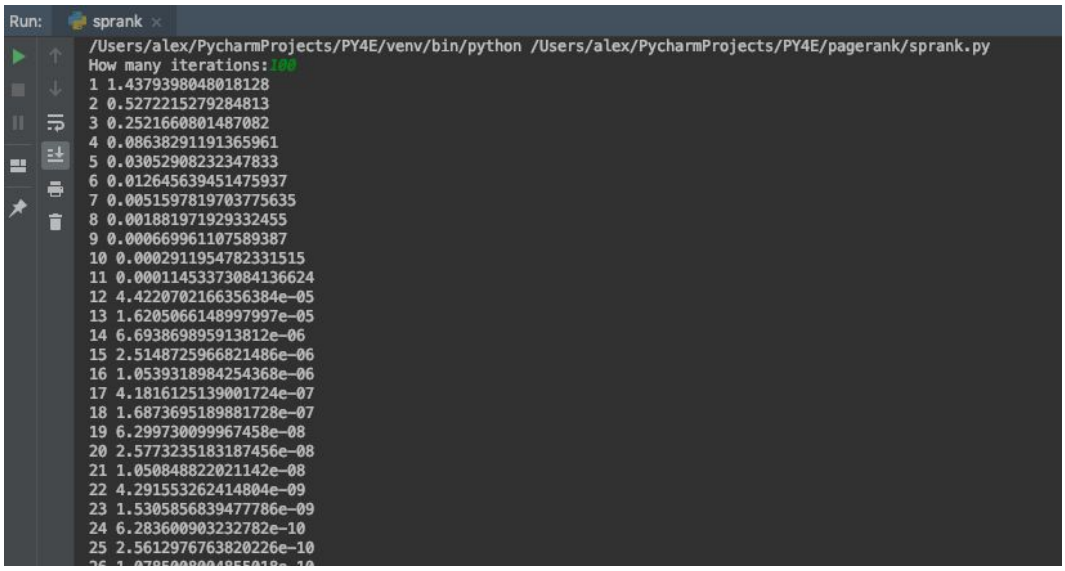
Spreset.py, run to reset the "spidering" process (and spider.sqlitedatabase) so we can "spider" another webpage for the assignment:

```
import sqlite3

conn = sqlite3.connect('spider.sqlite')
cur = conn.cursor()

cur.execute('''UPDATE Pages SET new_rank=1.0, old_rank=0.0''')
conn.commit()

cur.close()

print("All pages set to a rank of 1.0")

```

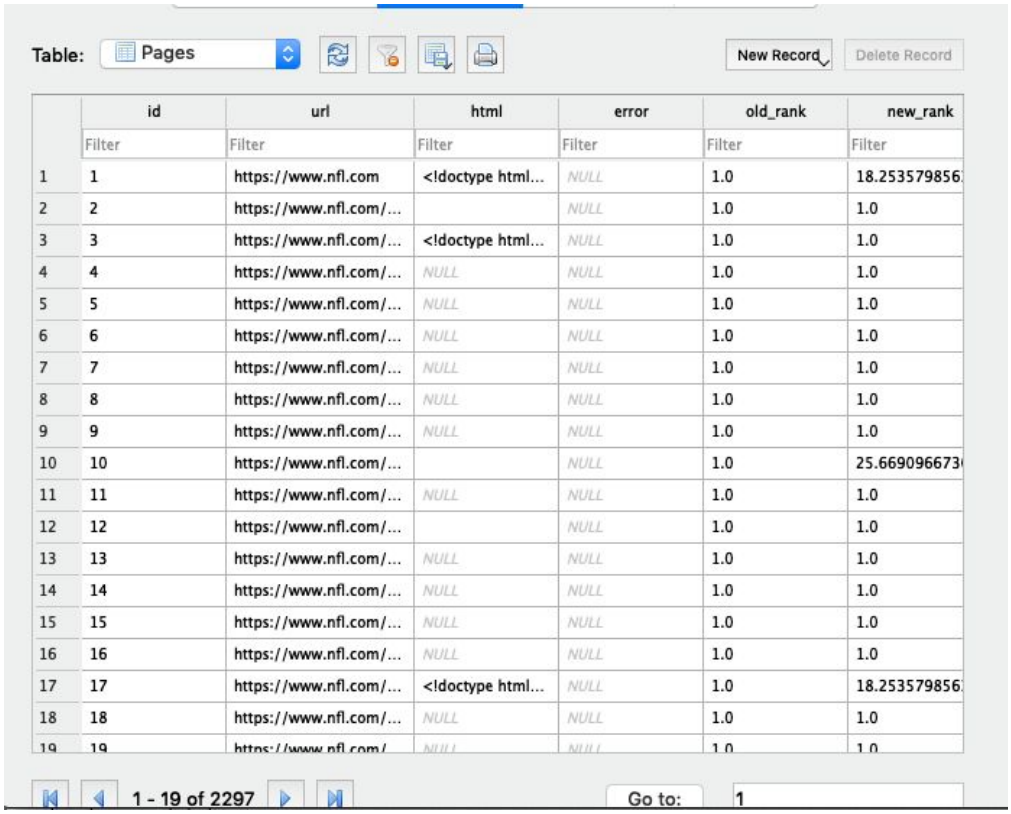
```
Run:  spreset
/Users/alex/PycharmProjects/PY4E/venv/bin/python /Users/alex/PycharmProjects/PY4E/pag
All pages set to a rank of 1.0

Process finished with exit code 0
```

Results from running spider.py, “spidering” 100 pages from https://www.nfl.com:

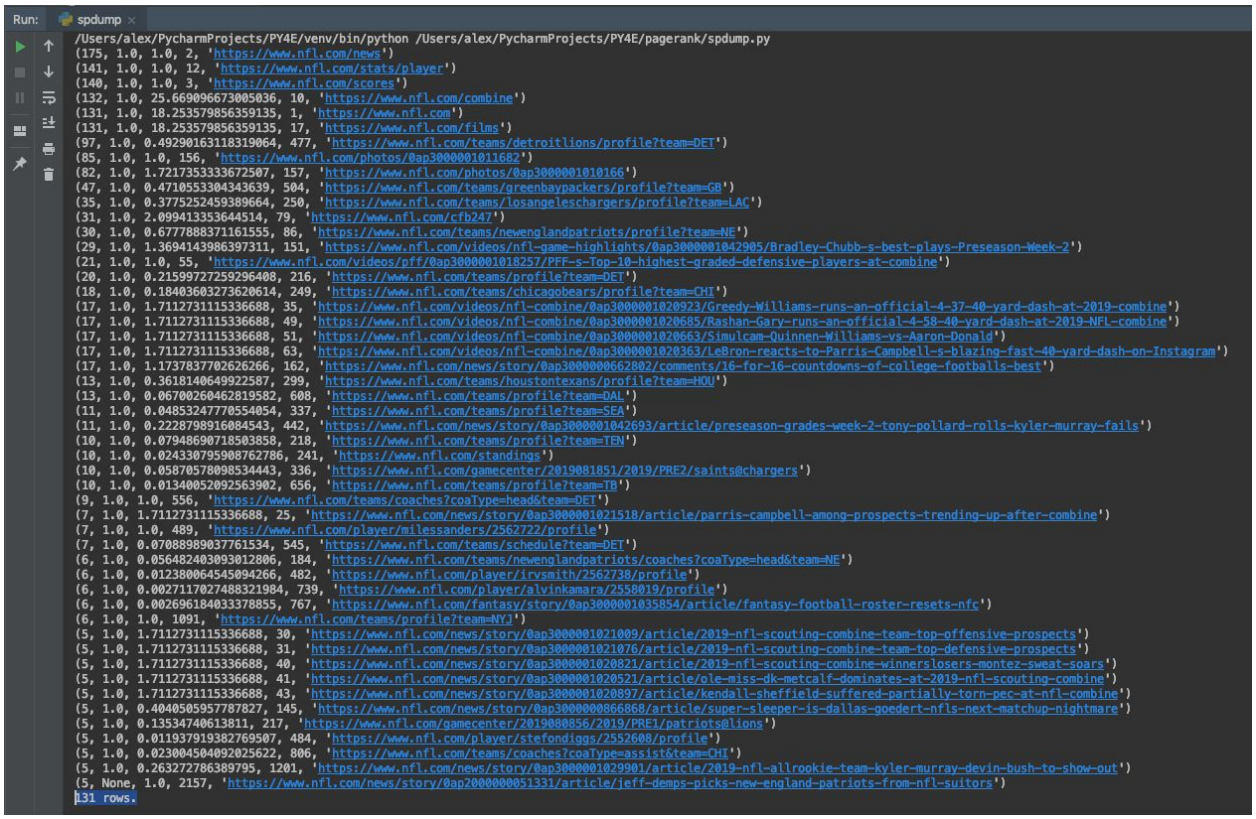
Results from running sprank.py for nlf.com (100 iterations to normalize the ranks as best as possible):



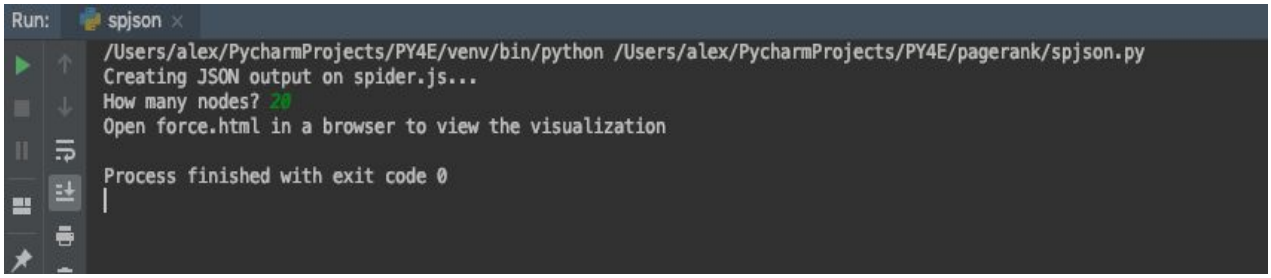
Spider.sqlite database, after running sprank.py for nfl.com, notice the rank values, nfl has more traffic than Dr. Chuck's website:



Spdump.py and output in terminal (Select statement w/JOIN to retrieve web page counts (frequencies), old rank, new rank, web page ID, URL of the web page, then prints it to the screen), screenshot needed for assignment (nfl.com):

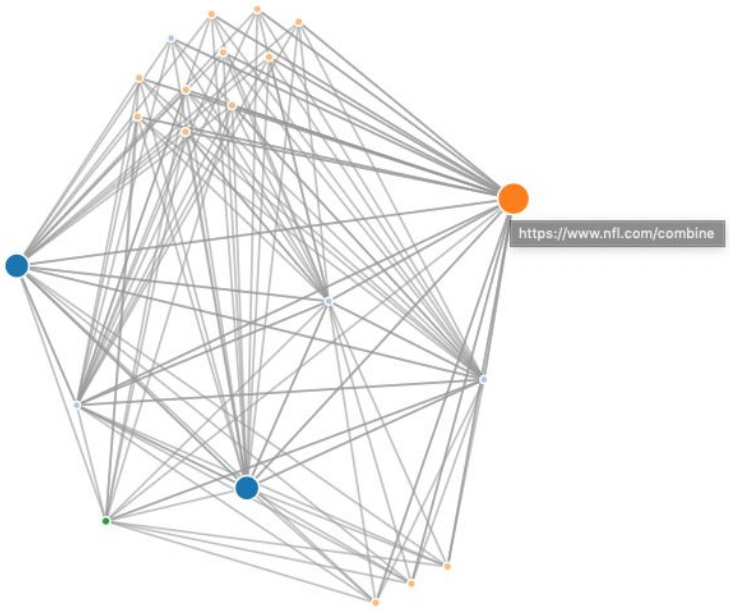


Spjson.py asking for 20 nodes from nfl.com:



Visualization for nfl.com, result from selecting the Chrome browser in force.html (screenshot need for assignment):

Starting url: https://www.nfl.com



If you don't see a chart above, check the JavaScript console. You may need to use a different browser.

F.2: Week 4 - Loading and Modeling Mail Data Assignment (Gmane/Mail pt.1):

Prompt for the assignment:

Welcome Alex Huerta from Capstone: Retrieving, Processing, and Visualizing Data with Python

Mailing List Data - Part I

In this assignment you will download some of the mailing list data from http://mbox.dr-chuck.net/ and run the data cleaning / modeling process and take some screen shots..

Don't take off points for little mistakes. If they seem to have done the assignment give them full credit. Feel free to make suggestions if there are small mistakes. Please keep your comments positive and useful. The course staff will assign the last 30% of the grade and it make take a few days to get graded.

Please Upload Your Submission:

A screen shot of your SQLiteBrowser showing messages downloaded from mbox.dr-chuck.net into the content.sqlite database

Choose File No file chosen

(Please use PNG or JPG files)

A screen shot of you running the gmodel.py application to produce the index.sqlite database.

Choose File No file chosen

(Please use PNG or JPG files)

A screen shot of your SQLiteBrowser showing messages in the index.sqlite database after the gmodel.py has executed.

Choose File No file chosen

(Please use PNG or JPG files)

A screen shot of you running the gbasic.py program to compute basic histogram data on the messages you have retrieved.

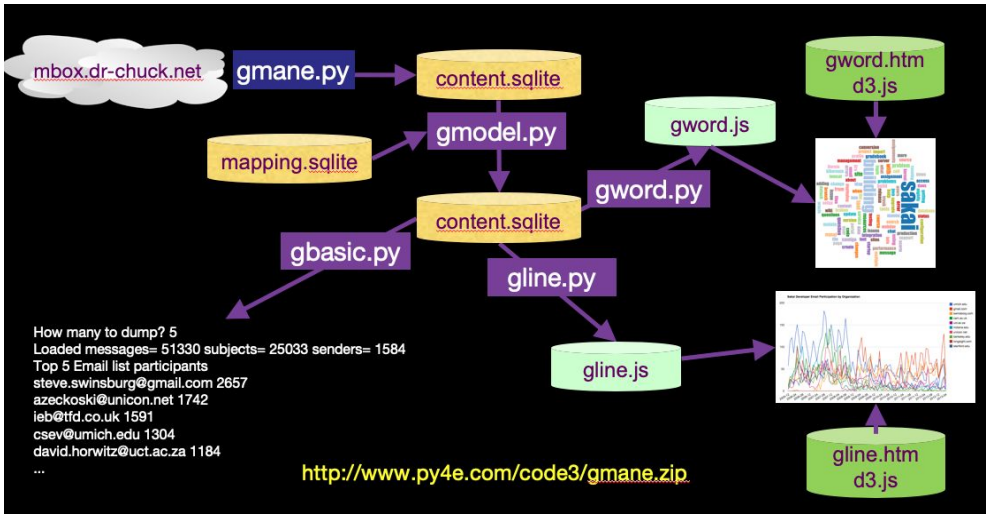
Choose File No file chosen

(Please use PNG or JPG files)

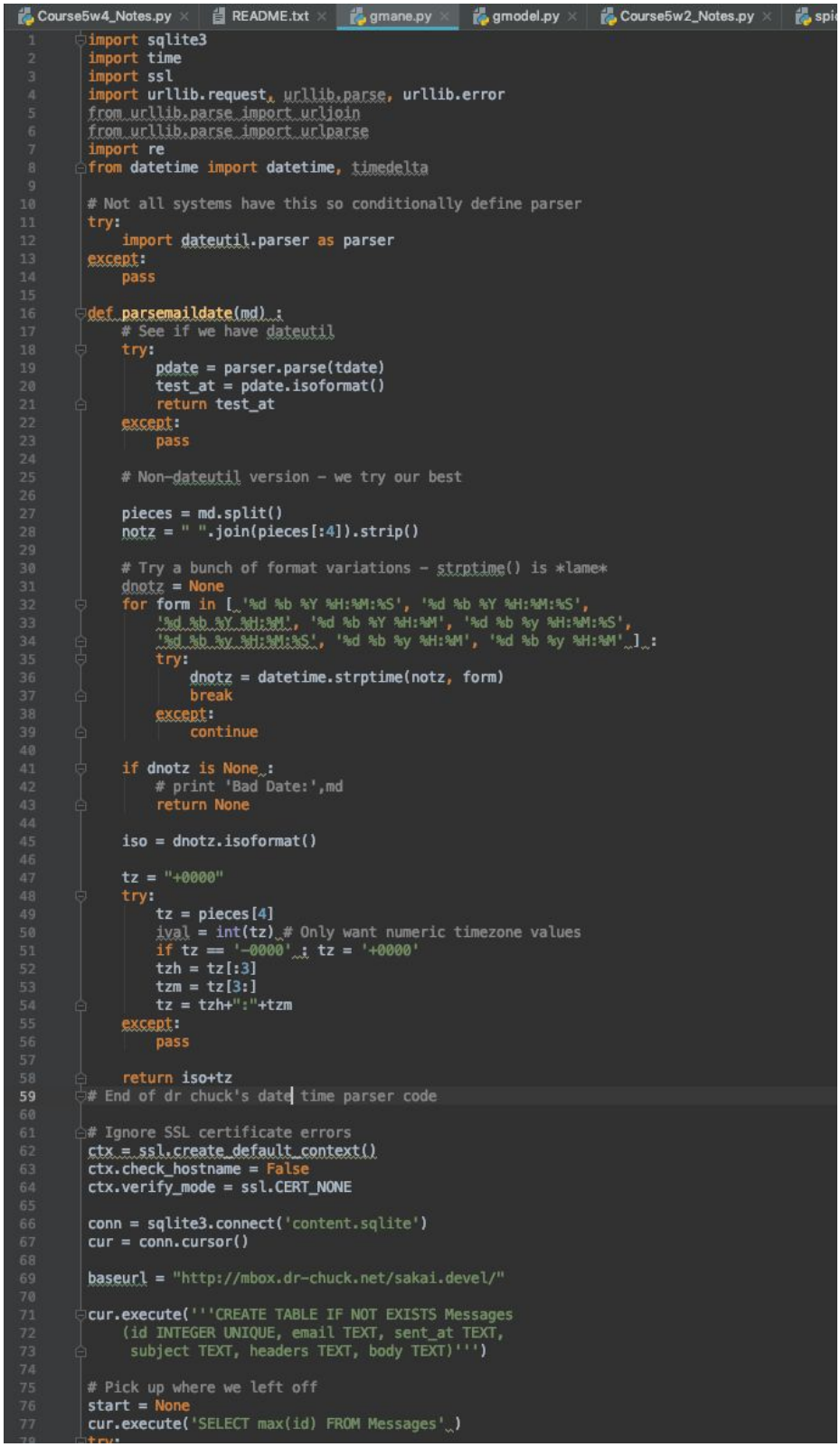
Enter optional comments below

Submit

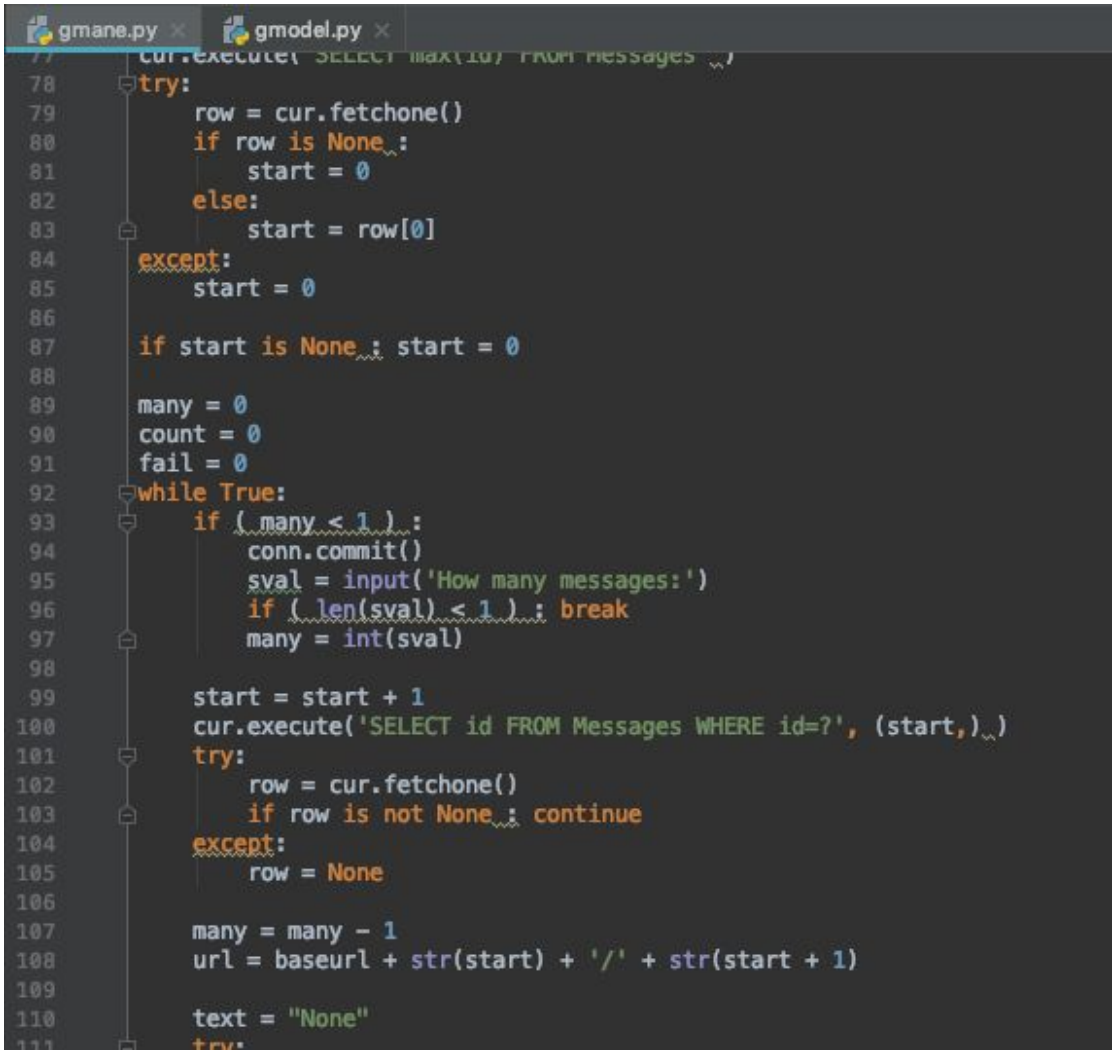
Diagram from the Ch.16 lecture slides, shows how each of the pages of code provided relate to each other, helps to explain what they do:



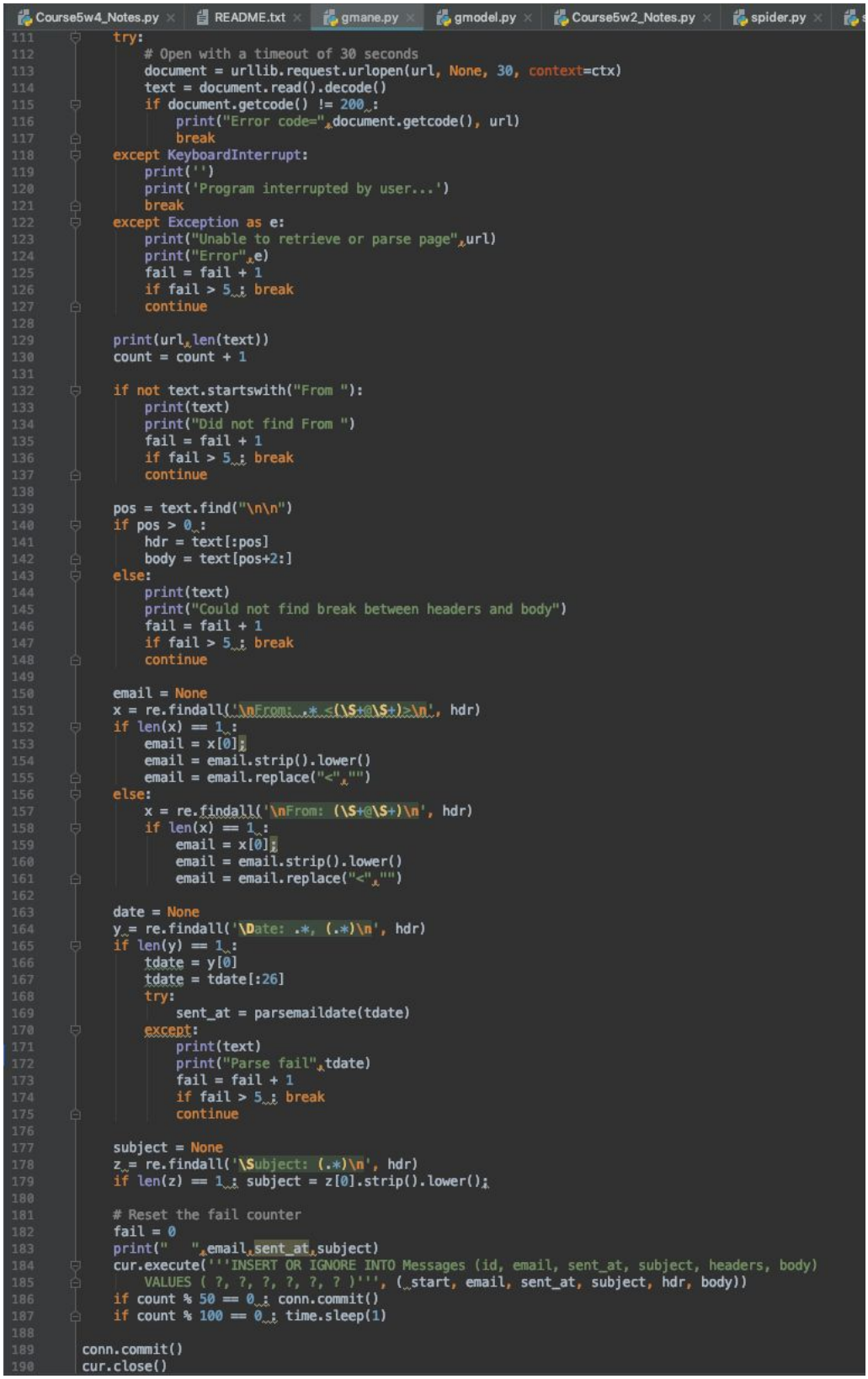
Gmane.py (pt.1):


Course5w4_Notes.py
README.txt
gmane.py
gmodel.py
Course5w2_Notes.py
import sqlite3
import time
import ssl
import urllib.request, urllib.parse, urllib.error
from urllib.parse import urljoin
from urllib.parse import urlparse
import re
from datetime import datetime, timedelta
# Not all systems have this so conditionally define parser
try:
import dateutil.parser as parser
except:
pass
def parsemaildate(md) :
# See if we have dateutil
try:
pdate = parser.parse(tdate)
test_at = pdate.isoformat()
return test_at
except:
pass
# Non-dateutil version - we try our best
pieces = md.split()
notz = " ".join(pieces[:4]).strip()
# Try a bunch of format variations - strptime() is *lame*
dnotz = None
for form in ['%d %b %Y %H:%M:%S', '%d %b %Y %H:%M:%S',
'%d %b %Y %H:%M', '%d %b %Y %H:%M', '%d %b %y %H:%M:%S',
'%d %b %y %H:%M:%S', '%d %b %y %H:%M', '%d %b %y %H:%M'] :
try:
dnotz = datetime.strptime(notz, form)
break
except:
continue
if dnotz is None :
print 'Bad Date:',md
return None
iso = dnotz.isoformat()
tz = "+0000"
try:
tz = pieces[4]
ival = int(tz) # Only want numeric timezone values
if tz == '-0000' : tz = '+0000'
tzh = tz[:3]
tzm = tz[3:]
tz = tzh+":"+tzm
except:
pass
return iso+tz
# End of dr chuck's date time parser code
# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE
conn = sqlite3.connect('content.sqlite')
cur = conn.cursor()
baseurl = "http://mbox.dr-chuck.net/sakai.devel/"
cur.execute('''CREATE TABLE IF NOT EXISTS Messages
(id INTEGER UNIQUE, email TEXT, sent_at TEXT,
subject TEXT, headers TEXT, body TEXT)''')
# Pick up where we left off
start = None
cur.execute('SELECT max(id) FROM Messages')
try:

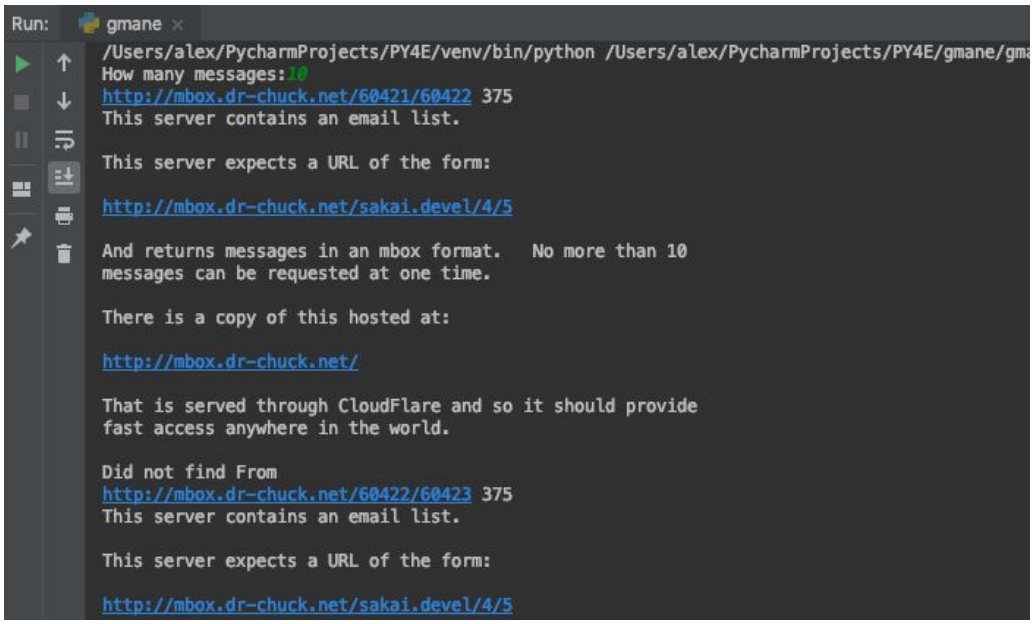
Gmane.py (pt.2):


gmane.py
gmodel.py
cur.execute(SELECT max(id) FROM Messages)
try:
row = cur.fetchone()
if row is None :
start = 0
else:
start = row[0]
except:
start = 0
if start is None : start = 0
many = 0
count = 0
fail = 0
while True:
if (many < 1) :
conn.commit()
sval = input('How many messages:')
if (len(sval) < 1) : break
many = int(sval)
start = start + 1
cur.execute('SELECT id FROM Messages WHERE id=?', (start,))
try:
row = cur.fetchone()
if row is not None : continue
except:
row = None
many = many - 1
url = baseurl + str(start) + '/' + str(start + 1)
text = "None"
try:

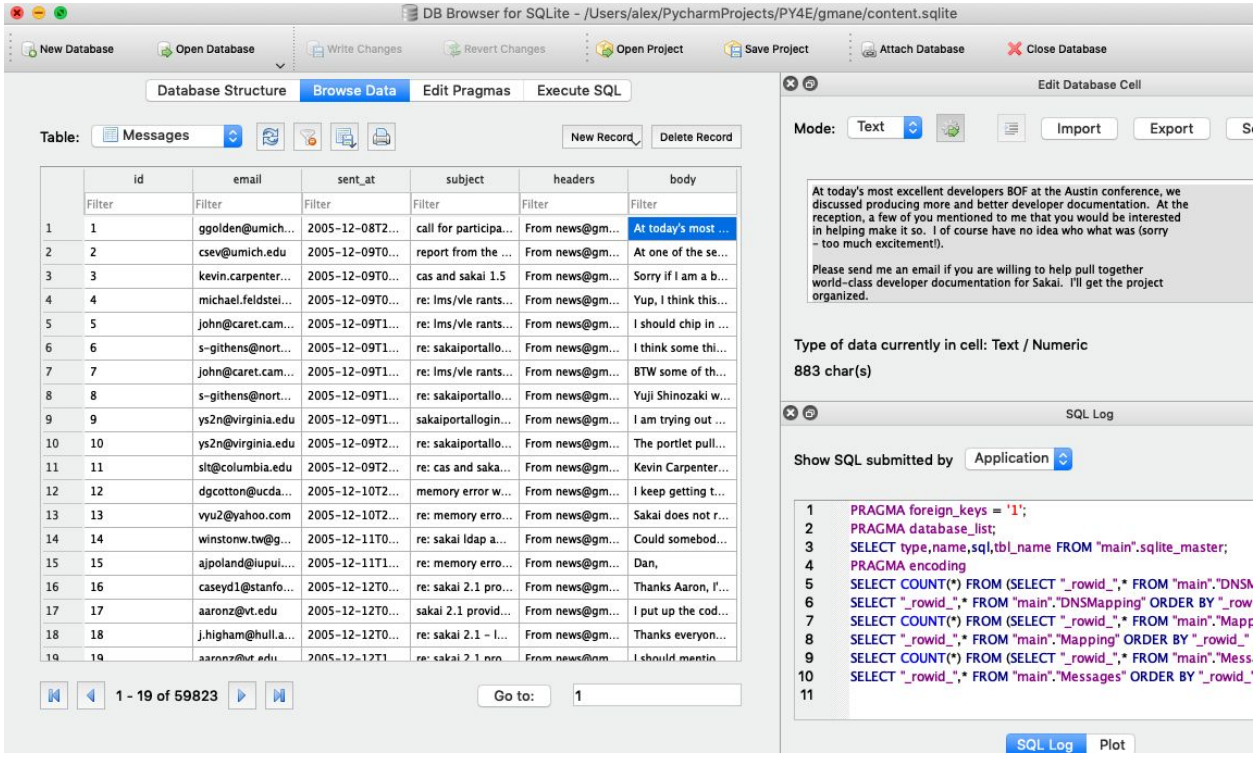
Gmane.py (pt.3):


Course5w4_Notes.py
README.txt
gmane.py
gmodel.py
Course5w2_Notes.py
spider.py
try:
# Open with a timeout of 30 seconds
document = urllib.request.urlopen(url, None, 30, context=ctx)
text = document.read().decode()
if document.getcode() != 200 :
print("Error code=",document.getcode(), url)
break
except KeyboardInterrupt:
print('')
print('Program interrupted by user...')
break
except Exception as e:
print("Unable to retrieve or parse page",url)
print("Error",e)
fail = fail + 1
if fail > 5 : break
continue
print(url,len(text))
count = count + 1
if not text.startswith("From "):
print(text)
print("Did not find From ")
fail = fail + 1
if fail > 5 : break
continue
pos = text.find("\n\n")
if pos > 0 :
hdr = text[:pos]
body = text[pos+2:]
else:
print(text)
print("Could not find break between headers and body")
fail = fail + 1
if fail > 5 : break
continue
email = None
x = re.findall('\nFrom: .* <(\S+@\S+)>\n', hdr)
if len(x) == 1 :
email = x[0];
email = email.strip().lower()
email = email.replace("<","")
else:
x = re.findall('\nFrom: (\S+@\S+)\n', hdr)
if len(x) == 1 :
email = x[0];
email = email.strip().lower()
email = email.replace("<","")
date = None
y = re.findall('\Date: .*, (.*)\n', hdr)
if len(y) == 1 :
tdate = y[0]
tdate = tdate[:26]
try:
sent_at = parsemaildate(tdate)
except:
print(text)
print("Parse fail",tdate)
fail = fail + 1
if fail > 5 : break
continue
subject = None
z = re.findall('\Subject: (.*)\n', hdr)
if len(z) == 1 : subject = z[0].strip().lower();
# Reset the fail counter
fail = 0
print(" ",email,sent_at,subject)
cur.execute('''INSERT OR IGNORE INTO Messages (id, email, sent_at, subject, headers, body)
VALUES (?, ?, ?, ?, ?, ?)''', (start, email, sent_at, subject, hdr, body))
if count % 50 == 0 : conn.commit()
if count % 100 == 0 : time.sleep(1)
conn.commit()
cur.close()

Results from running gmane.py, spidering email messages and creating content.sqlite:

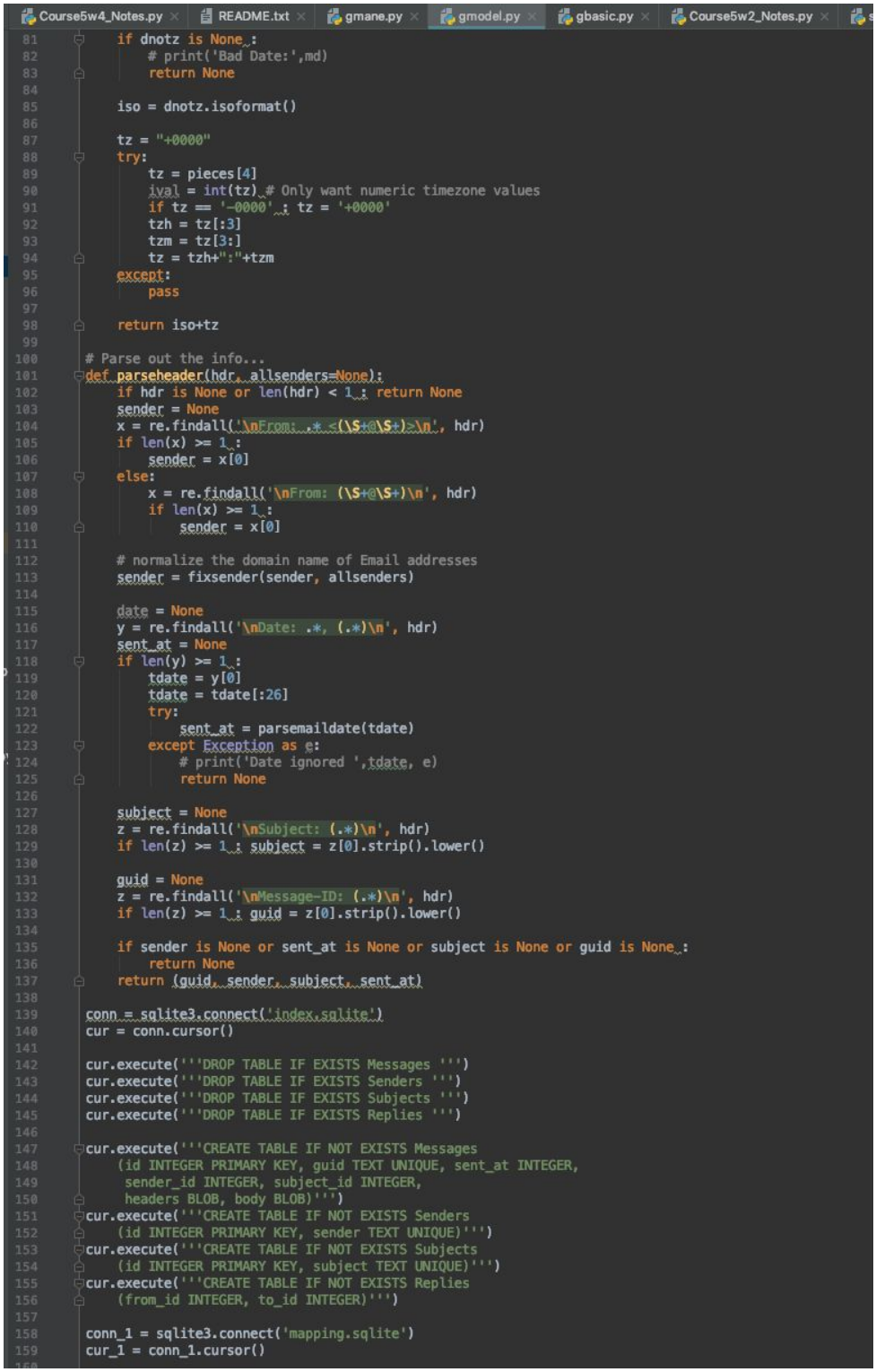


Content.sqlite database (raw data), created by running gmane.py ("spidering" 100 messages), screenshot needed for assignment:



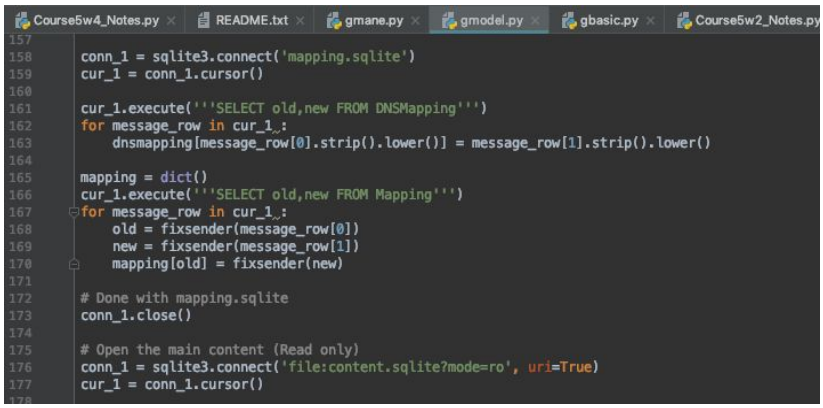
Gmodel.py (pt.1):

Gmodel.py (pt.2):




```
    if dnotz is None:
        # print('Bad Date:',md)
        return None

    iso = dnotz.isoformat()

    tz = "+0000"
    try:
        tz = pieces[4]
        ival = int(tz) # Only want numeric timezone values
        if tz == '-0000' : tz = '+0000'
        tzh = tz[:3]
        tzm = tz[3:]
        tz = tzh+":"+tzm
    except:
        pass

    return iso+tz

# Parse out the info...
def parseheader(hdr, allsenders=None):
    if hdr is None or len(hdr) < 1 : return None
    sender = None
    x = re.findall('\nFrom: .* <(\S+@\S+)>\n', hdr)
    if len(x) >= 1 :
        sender = x[0]
    else:
        x = re.findall('\nFrom: (\S+@\S+)\n', hdr)
        if len(x) >= 1 :
            sender = x[0]

    # normalize the domain name of Email addresses
    sender = fixsender(sender, allsenders)

    date = None
    y = re.findall('\nDate: .*, (.*)\n', hdr)
    sent_at = None
    if len(y) >= 1 :
        tdate = y[0]
        tdate = tdate[:26]
        try:
            sent_at = parsemaildate(tdate)
        except Exception as e:
            # print('Date ignored ',tdate, e)
            return None

    subject = None
    z = re.findall('\nSubject: (.*)\n', hdr)
    if len(z) >= 1 : subject = z[0].strip().lower()

    guid = None
    z = re.findall('\nMessage-ID: (.*)\n', hdr)
    if len(z) >= 1 : guid = z[0].strip().lower()

    if sender is None or sent_at is None or subject is None or guid is None :
        return None
    return (guid, sender, subject, sent_at)

conn = sqlite3.connect('index.sqlite')
cur = conn.cursor()

cur.execute('''DROP TABLE IF EXISTS Messages ''')
cur.execute('''DROP TABLE IF EXISTS Senders ''')
cur.execute('''DROP TABLE IF EXISTS Subjects ''')
cur.execute('''DROP TABLE IF EXISTS Replies ''')

cur.execute('''CREATE TABLE IF NOT EXISTS Messages
    (id INTEGER PRIMARY KEY, guid TEXT UNIQUE, sent_at INTEGER,
     sender_id INTEGER, subject_id INTEGER,
     headers BLOB, body BLOB)''')
cur.execute('''CREATE TABLE IF NOT EXISTS Senders
    (id INTEGER PRIMARY KEY, sender TEXT UNIQUE)''')
cur.execute('''CREATE TABLE IF NOT EXISTS Subjects
    (id INTEGER PRIMARY KEY, subject TEXT UNIQUE)''')
cur.execute('''CREATE TABLE IF NOT EXISTS Replies
    (from_id INTEGER, to_id INTEGER)''')

conn_1 = sqlite3.connect('mapping.sqlite')
cur_1 = conn_1.cursor()
```

Gmodel.py (pt.3):

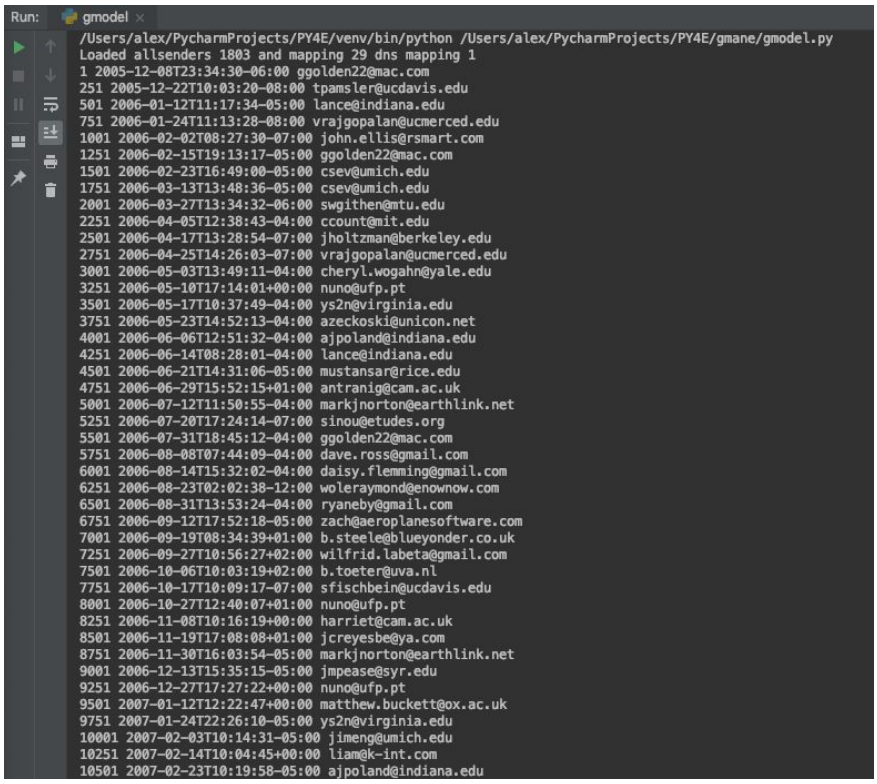


```
conn_1 = sqlite3.connect('mapping.sqlite')
cur_1 = conn_1.cursor()

cur_1.execute('''SELECT old,new FROM DNSMapping''')
for message_row in cur_1:
    dnsmapping[message_row[0].strip().lower()] = message_row[1].strip().lower()

mapping = dict()
cur_1.execute('''SELECT old,new FROM Mapping''')
for message_row in cur_1:
    old = fixsender(message_row[0])
    new = fixsender(message_row[1])
    mapping[old] = fixsender(new)

# Done with mapping.sqlite
conn_1.close()

# Open the main content (Read only)
conn_1 = sqlite3.connect('file:content.sqlite?mode=ro', uri=True)
cur_1 = conn_1.cursor()
```

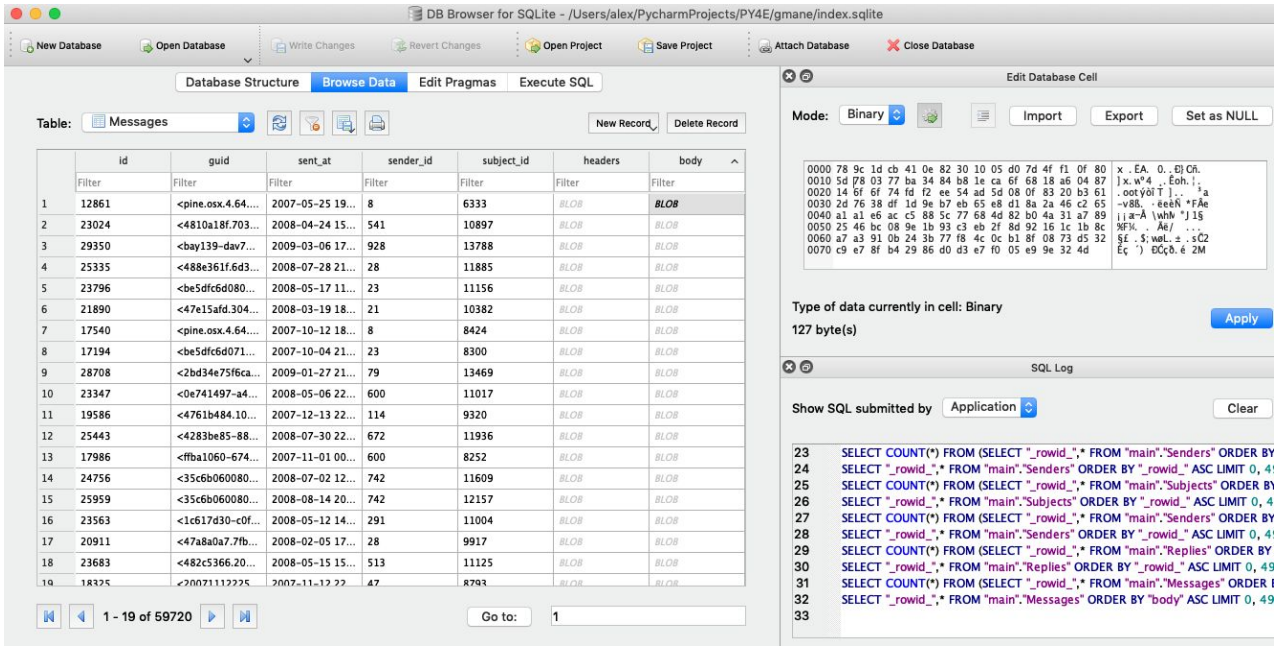
Gmodel.py (pt.4):

```
cur_1 = conn_1.cursor()

allsenders = list()
cur_1.execute('''SELECT email FROM Messages''')
for message_row in cur_1:
    sender = fixsender(message_row[0])
    if sender is None: continue
    if 'gmane.org' in sender: continue
    if sender in allsenders: continue
    allsenders.append(sender)

print("Loaded allsenders",len(allsenders),"and mapping",len(mapping),"dns mapping",len(dnsmapping))

cur_1.execute('''SELECT headers, body, sent_at
    FROM Messages ORDER BY sent_at''')

senders = dict()
subjects = dict()
guids = dict()

count = 0

for message_row in cur_1:
    hdr = message_row[0]
    parsed = parseheader(hdr, allsenders)
    if parsed is None: continue
    (guid, sender, subject, sent_at) = parsed

    # Apply the sender mapping
    sender = mapping.get(sender,sender)

    count = count + 1
    if count % 250 == 1: print(count,sent_at, sender)
    # print(guid, sender, subject, sent_at)

    if 'gmane.org' in sender:
        print("Error in sender ===", sender)

    sender_id = senders.get(sender,None)
    subject_id = subjects.get(subject,None)
    guid_id = guids.get(guid,None)

    if sender_id is None:
        cur.execute('INSERT OR IGNORE INTO Senders (sender) VALUES ( ? )', (sender, ))
        conn.commit()
        cur.execute('SELECT id FROM Senders WHERE sender=? LIMIT 1', (sender, ))
        try:
            row = cur.fetchone()
            sender_id = row[0]
            senders[sender] = sender_id
        except:
            print('Could not retrieve sender id',sender)
            break
    if subject_id is None:
        cur.execute('INSERT OR IGNORE INTO Subjects (subject) VALUES ( ? )', (subject, ))
        conn.commit()
        cur.execute('SELECT id FROM Subjects WHERE subject=? LIMIT 1', (subject, ))
        try:
            row = cur.fetchone()
            subject_id = row[0]
            subjects[subject] = subject_id
        except:
            print('Could not retrieve subject id',subject)
            break
    # print(sender_id, subject_id)
    cur.execute('INSERT OR IGNORE INTO Messages (guid,sender_id,subject_id,sent_at,headers,body) VALUES ( ?,?,?,datetime(?),?,? )',
            ( guid, sender_id, subject_id, sent_at,
            zlib.compress(message_row[0].encode()), zlib.compress(message_row[1].encode())))
    conn.commit()
    cur.execute('SELECT id FROM Messages WHERE guid=? LIMIT 1', (guid, ))
    try:
        row = cur.fetchone()
        message_id = row[0]
        guids[guid] = message_id
    except:
        print('Could not retrieve guid id',guid)
        break

cur.close()
cur_1.close()
```

Results from running gmodel.py, creates index.sqlite, prints data (guid, date/time message sent, and the sender's email) thats being inserted into database (screenshot needed for assignment):



Index.sqlite (cleaned-up data)m created by gmodel.py (screenshot needed for assignment):



F.3: Week 6 - Visualizing Email Data Assignment (Gmane/Mail pt.2):

Prompt for the assignment:

Welcome Alex Huerta from Capstone: Retrieving, Processing, and Visualizing Data with Python

Mailing List Data - Part II

In this assignment you will visualize the mailing list data you have downloaded from http://mbox.dr-chuck.net/ and take some screen shots. Important: You do not have to download all of the data - 3 months is 1700 messages, 6 mths 3200 and 1 yr 9000. Do not use low counts like 1, 20, 100, 300. You can run gmane multiple times to download more messages. It is completely acceptable to visualize a small subset of the data in the gbasic screenshot.

Don't take off points for little mistakes. If they seem to have done the assignment give them full credit. Feel free to make suggestions if there are small mistakes. Please keep your comments positive and useful. The course staff will assign the last 30% of the grade and it make take a few days to get graded. - For students in mainland China that can't access the Google API for the timeline, take a screenshot of gline.js open in your editor showing the numbers, dates of messages at the top. Please add a note to your assignment.

Please Upload Your Submission:

A screen shot of you running the gbasic.py program to compute basic histogram data on the messages you have retrieved.

Choose File No file chosen

(Please use PNG or JPG files)

A screen shot of word cloud visualization for the messages you have retrieved.

Choose File No file chosen

(Please use PNG or JPG files)

A screen shot of time line visualization for the messages you have retrieved.

Choose File No file chosen

(Please use PNG or JPG files)

Optional Challenge: Change the gline.py program to show the message count by year instead of by month and take a screen shot of the by year visualization. You can switch from a by-month to a by-year visualization by changing only a few lines in gline.py. The puzzle is to figure out the smallest change to accomplish the change. If you do not want to do this optional challenge - just upload the above image a second time.

Choose File No file chosen

(Please use PNG or JPG files)

Enter optional comments below

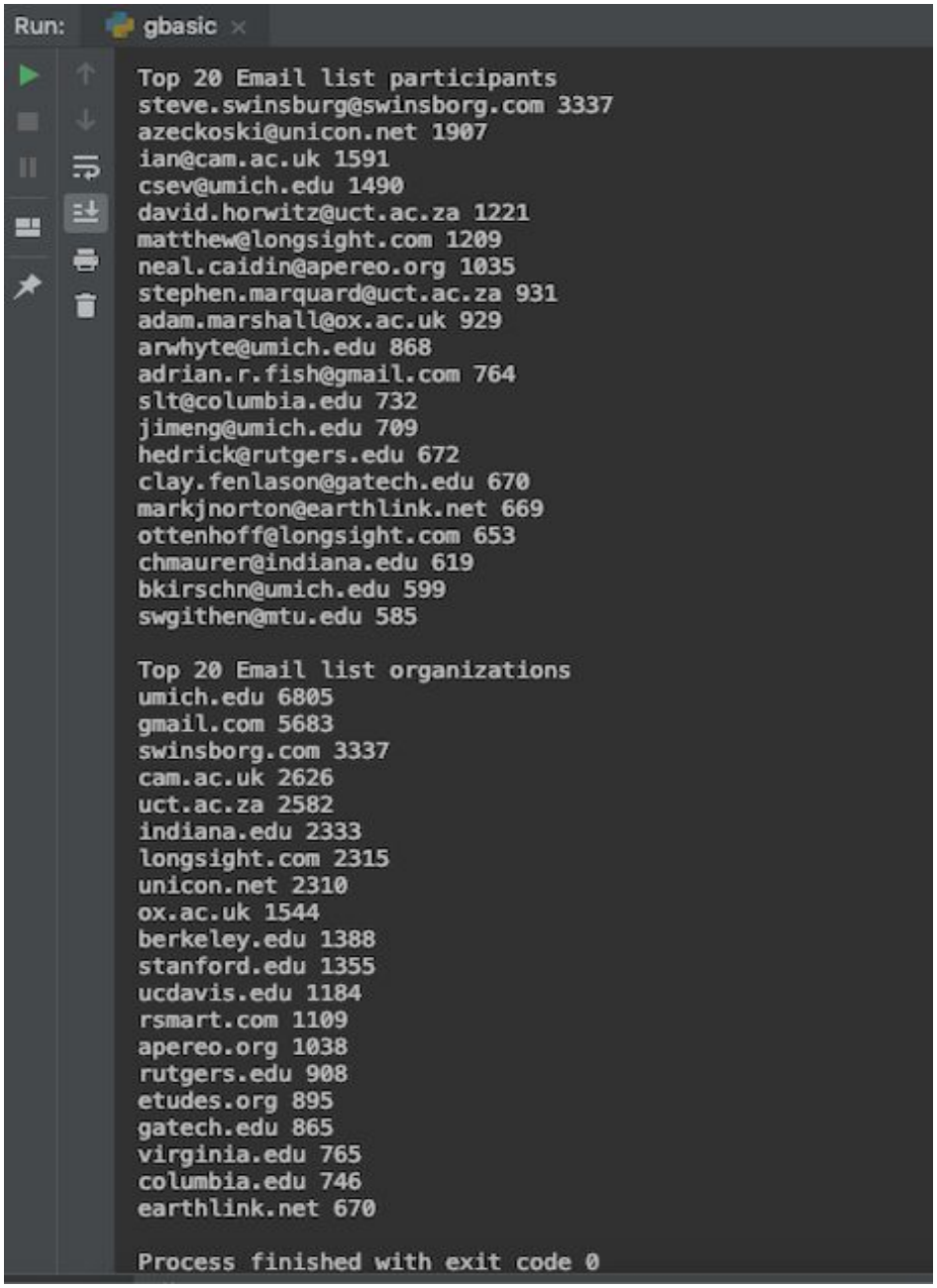
Submit

Gbasic.py:

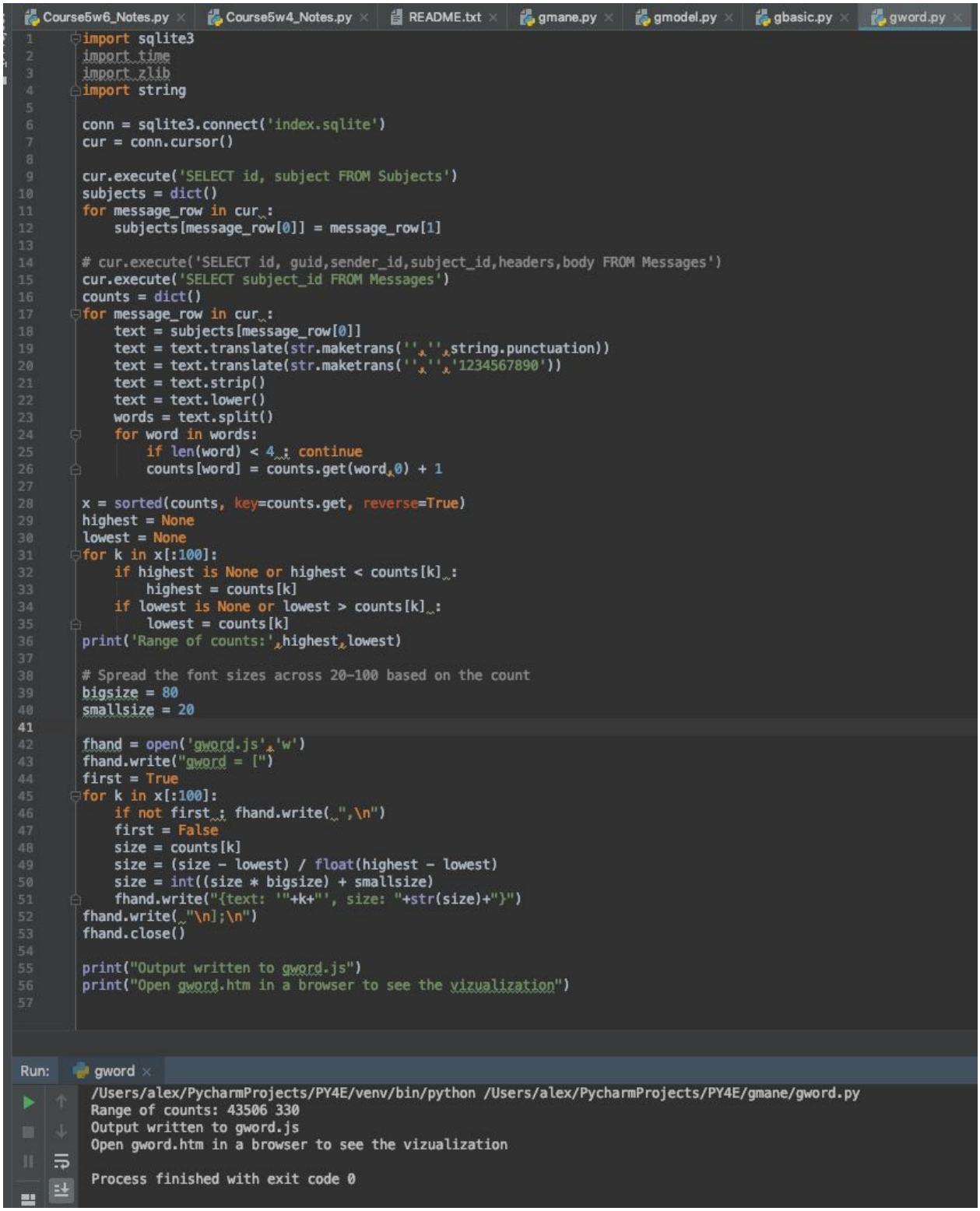
gline.py | gline.js | gline.htm | gbasic.py | gword.py | gword.js | gword.htm

```
import sqlite3
import time
import zlib
# gets basic data

howmany = int(input("How many to dump? "))

conn = sqlite3.connect('index.sqlite')
cur = conn.cursor()

cur.execute('SELECT id, sender FROM Senders')
senders = dict()
for message_row in cur:
    senders[message_row[0]] = message_row[1]

cur.execute('SELECT id, subject FROM Subjects')
subjects = dict()
for message_row in cur:
    subjects[message_row[0]] = message_row[1]

# cur.execute('SELECT id, guid,sender_id,subject_id,headers,body FROM Messages')
cur.execute('SELECT id, guid,sender_id,subject_id,sent_at FROM Messages')
messages = dict()
for message_row in cur:
    messages[message_row[0]] = (message_row[1],message_row[2],message_row[3],message_row[4])

print("Loaded messages=",len(messages),"subjects=",len(subjects),"senders=",len(senders))

sendcounts = dict()
sendorgs = dict()
for (message_id, message) in list(messages.items()):
    sender = message[1]
    sendcounts[sender] = sendcounts.get(sender,0) + 1
    pieces = senders[sender].split("@")
    if len(pieces) != 2 : continue
    dns = pieces[1]
    sendorgs[dns] = sendorgs.get(dns,0) + 1

print('')
print('Top',howmany,'Email list participants')

x = sorted(sendcounts, key=sendcounts.get, reverse=True)
for k in x[:howmany]:
    print(senders[k], sendcounts[k])
    if sendcounts[k] < 10 : break

print('')
print('Top',howmany,'Email list organizations')

x = sorted(sendorgs, key=sendorgs.get, reverse=True)
for k in x[:howmany]:
    print(k, sendorgs[k])
    if sendorgs[k] < 10 : break

```

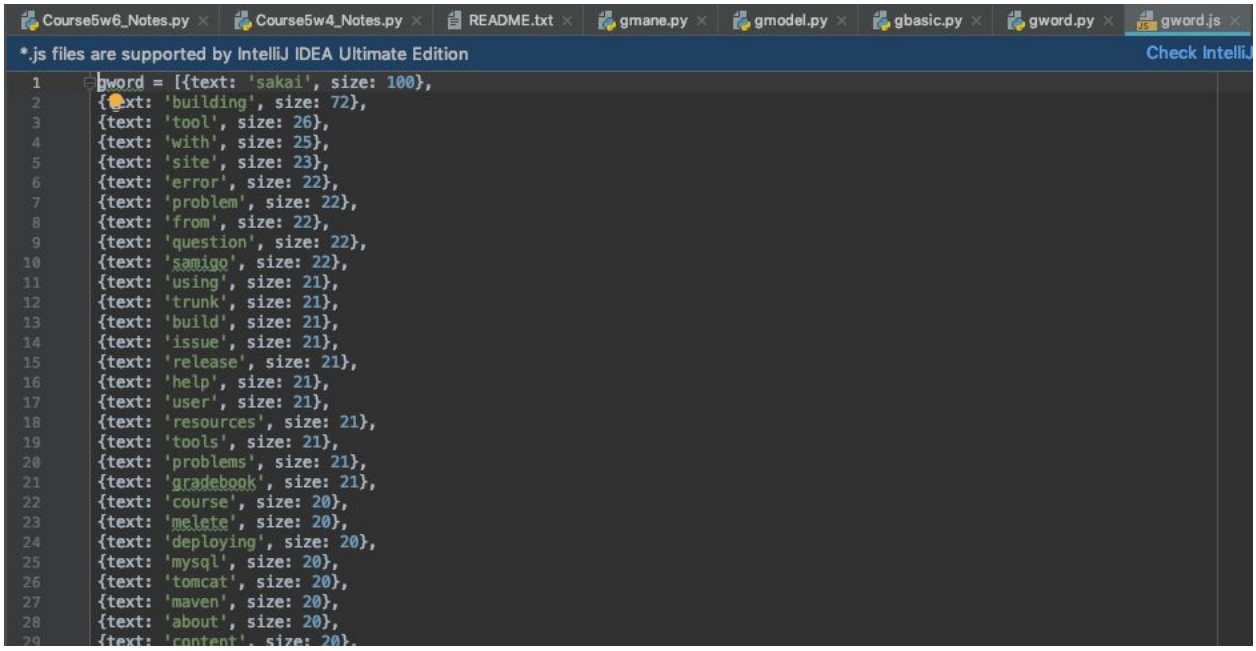
Results from running gbasic.py, top 20 email list participants & organizations (screenshot need for week 4 assignment and this assignment):



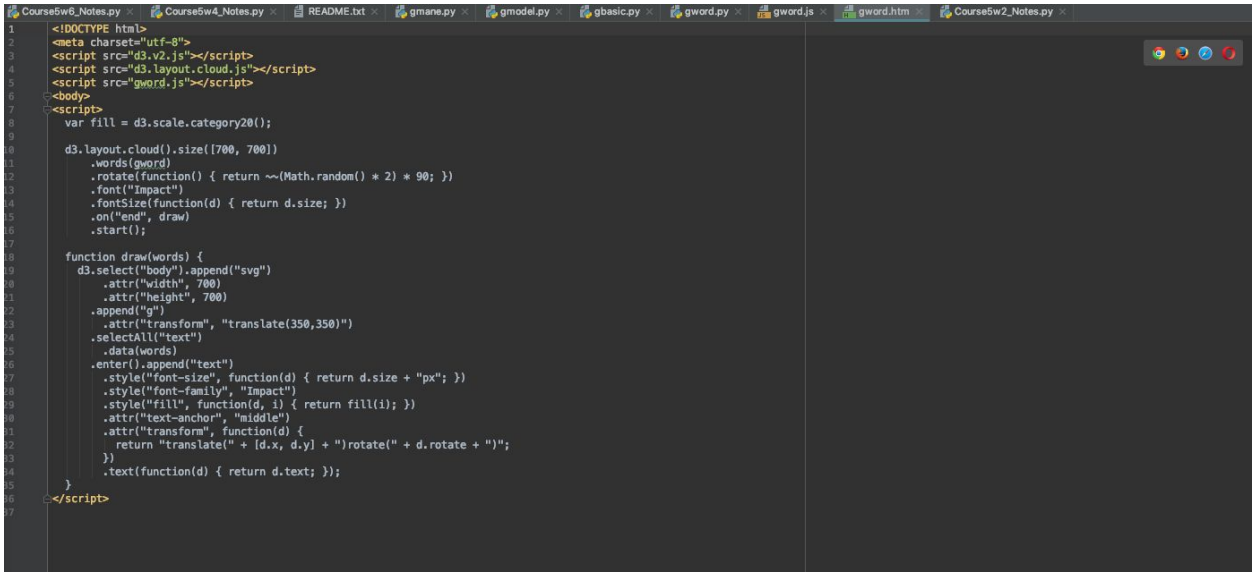
Gword.py, creates gword.js:



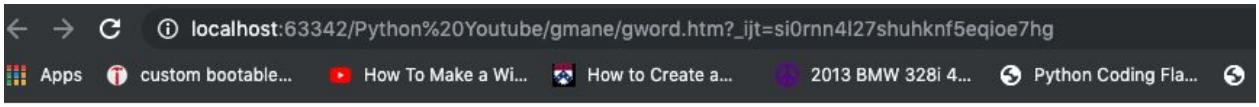
Gword.js (dimension and design info for d3 visualization):



Gword.htm (link to Word Cloud Visualization):



Word Cloud Visualization (screenshot needed for assignment):

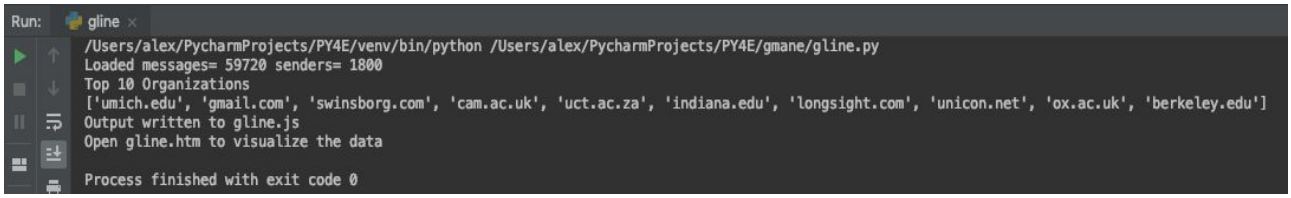


Gline.py:

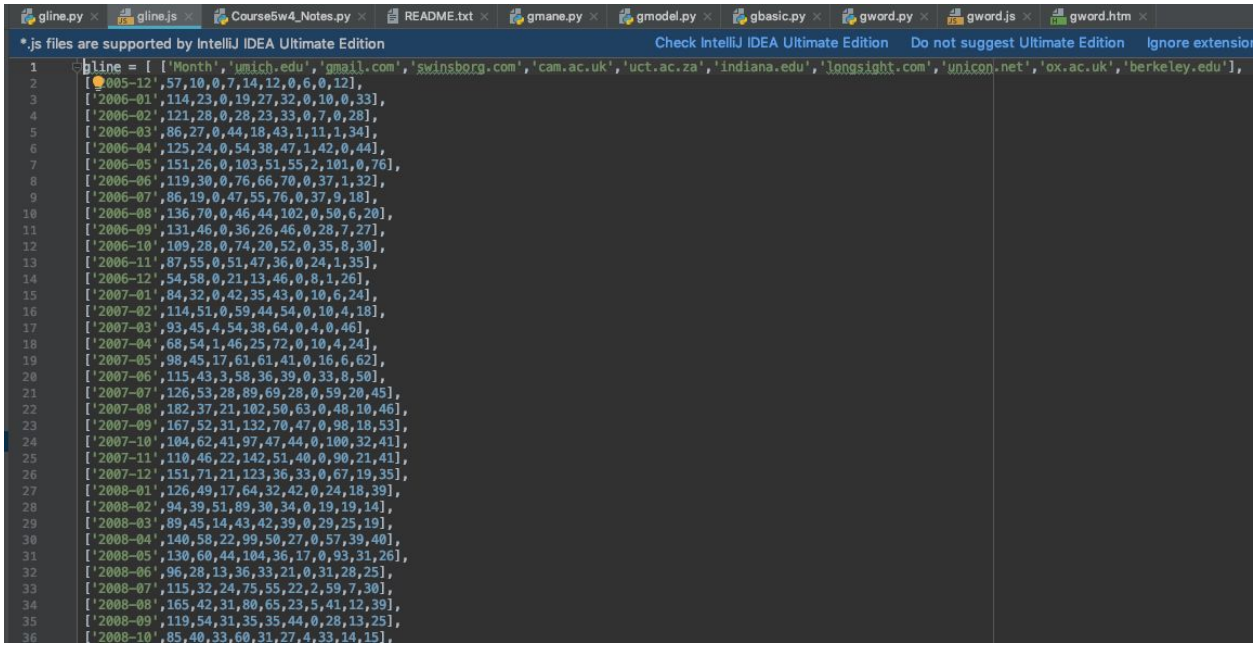
gline.py | Course5w6_Notes.py | Course5w4_Notes.py | README.txt | gmane.py | gmodel.py | gbasic.py

```
import sqlite3
import time
import zlib

conn = sqlite3.connect('index.sqlite')
cur = conn.cursor()

cur.execute('SELECT id, sender FROM Senders')
senders = dict()
for message_row in cur :
    senders[message_row[0]] = message_row[1]

cur.execute('SELECT id, guid,sender_id,subject_id,sent_at FROM Messages')
messages = dict()
for message_row in cur :
    messages[message_row[0]] = (message_row[1],message_row[2],message_row[3],message_row[4])

print("Loaded messages=",len(messages),"senders=",len(senders))

sendorgs = dict()
for (message_id, message) in list(messages.items()):
    sender = message[1]
    pieces = senders[sender].split("@")
    if len(pieces) != 2 : continue
    dns = pieces[1]
    sendorgs[dns] = sendorgs.get(dns,0) + 1

# pick the top schools
orgs = sorted(sendorgs, key=sendorgs.get, reverse=True)
orgs = orgs[:10]
print("Top 10 Organizations")
print(orgs)

counts = dict()
months = list()
# cur.execute('SELECT id, guid,sender_id,subject_id,sent_at FROM Messages')
for (message_id, message) in list(messages.items()):
    sender = message[1]
    pieces = senders[sender].split("@")
    if len(pieces) != 2 : continue
    dns = pieces[1]
    if dns not in orgs : continue
    month = message[3][:7]
    if month not in months : months.append(month)
    key = (month, dns)
    counts[key] = counts.get(key,0) + 1

months.sort()
# print counts
# print months

fhand = open('gline.js','w')
fhand.write("gline = [ ['Month'")
for org in orgs:
    fhand.write(",'"+org+"'")
fhand.write("]")

for month in months:
    fhand.write(",\n['"+month+"'")
    for org in orgs:
        key = (month, org)
        val = counts.get(key,0)
        fhand.write(","+str(val))
    fhand.write("]");

fhand.write("\n];\n")
fhand.close()

print("Output written to gline.js")
print("Open gline.htm to visualize the data")
```

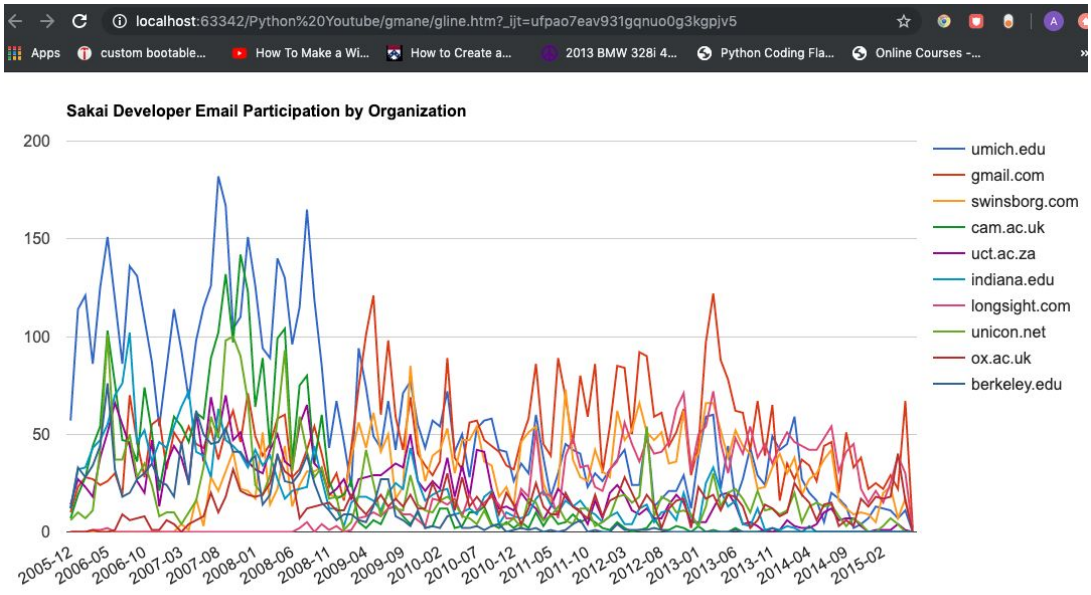
Results from running gline.py (top 10 organizations):



Gline.js (dimension and design info for d3 visualization):



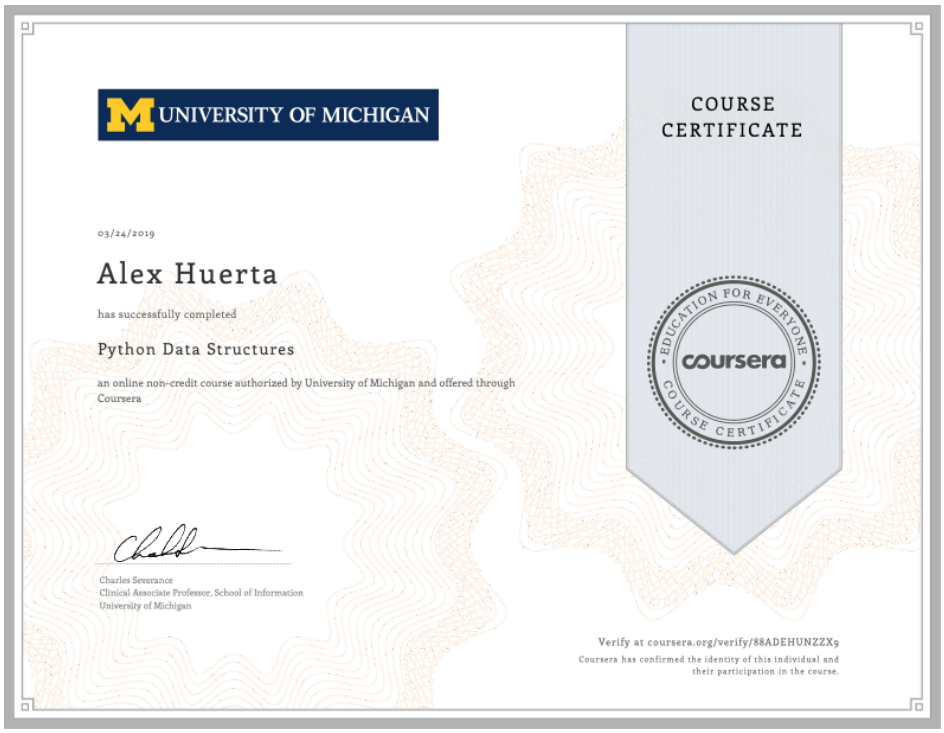
Gline.htm (link to Time-Line Visualization), screenshot needed for assignment:



G. Course Certificates

G.1: Course 1

G.2: Course 2



G.3: Course 3

G.4: Course 4

G.5: Course 5

UNIVERSITY OF MICHIGAN

COURSE CERTIFICATE

08/24/2019

Alex Huerta

has successfully completed

Capstone: Retrieving, Processing, and Visualizing Data with Python

an online non-credit course authorized by University of Michigan and offered through Coursera

EDUCATION FOR EVERYONE
coursera
COURSE CERTIFICATE

Charles Severance
Clinical Professor, School of Information
University of Michigan

Verify at coursera.org/verify/ETJVC5YLG234
Coursera has confirmed the identity of this individual and their participation in the course.