Anthony Clark
Ryan Farber
Jennifer Anderson

CS 419
Final Report
Due 12/11/14

Group 8 CS419 Software Projects

1. A short introduction to the project.
    - Executive summary of what you did.

      We created a program to handle advising appointment requests that creates an outlook calendar item for an advisor, while also allowing them to modify the appointments with a client interface.
    - What is its importance?

      It is important that professors have a convenient way to allow students to schedule appointments with them that they can accept or cancel. Many professors probably get lots of email to sort through on a daily basis, so creating a system which allows students to contact advisors and get a meeting on schedule allows students easier access to professors. Also, since the program uses Outlook meeting requests, the professors will be able to see at a glance what their schedule looks like and whether a student's request conflicts with their schedule. Finally, managing these appointments has been streamlined into one interface where an advisor can quickly, and easily view all scheduled appointments and cancel them, consequently sending an email which also removes it from their outlook calendar.
    - Who was/were your client(s)?

      Our clients are both professors at Oregon State University who will use this program to allow students to request appointments, and students at Oregon State who will request appointments through this system.
    - What were their roles?

The professors' roles is to check his/her email through Outlook or Linux and accept or cancel students' requests for meetings. They may also view the emails that are sent to them for quick information to see new meetings. In addition, they are able to remove any appointments from their calendar that they would like using the curses command line interface.

The students' roles are to request meetings in the correct format from professors via email by sending an email from the webform. It is an assumption of our project that these web forms will require data in the same format as the examples provided.

2. How did the project change since the original design document?

   The email sender program is the same as the original design document. Its sends an email in the proper format and allows for testing our email parser. However, we had originally thought that this program would be integrated into our final project, before realizing that we of course needed the procmail filter to pipe directly into our email parser program, instead of using the email sender and catcher. This was of little trouble as the conversion was easily implemented. Further more, in the initial design document, we wrote that procmail filter was supposed to change the plain text email to an Outlook email. However, our procmail file simply filters out relevant emails and sends them to the email parser to translate into an Outlook appointment. The procmail filter also doesn't update the database.

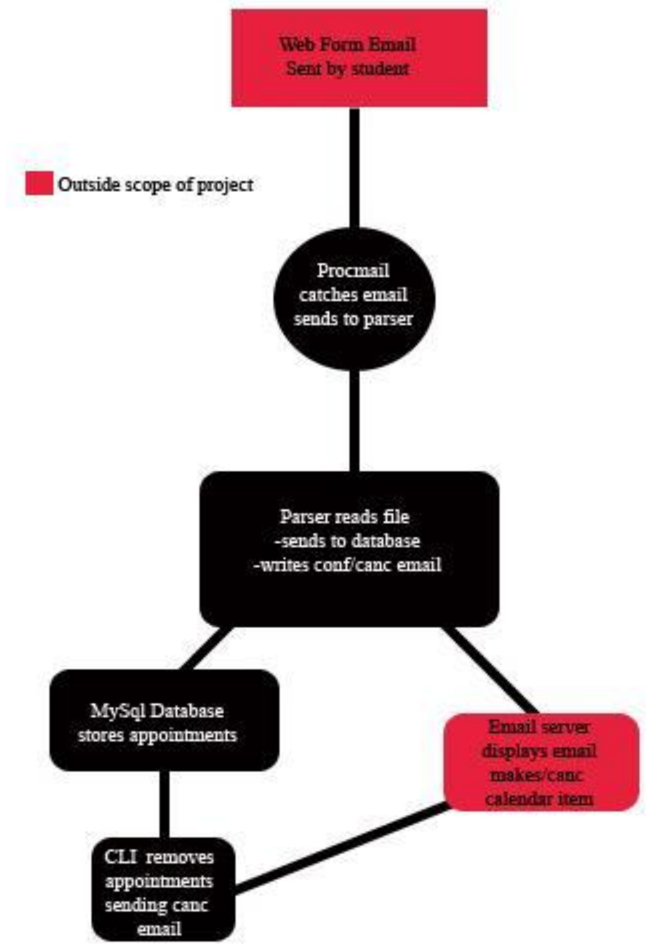
   Aside from these technical aspects, we differed significantly in our projected milestones and where we ended up completing different phases of the project.

3. Project documentation.
   - How does your program work?

     Our program works starting at the point where a student populates and sends a web form, which will send an email to the advisors .engr account. At this point, their procmail filter will catch the email, and filter the entirety of it into our parseEmail.py file, which is determined by a path specified within the procmail filter. The email is then converted into a large string where it is split apart, and has relevant information stored into variables. These variables are

updated to our database to store the appointment, and then later used to recreate the email that was initially sent to the advisor's onid or other specified account, with a calendar item now attached. This calendar item will then be sent along with the data to the advisor and depending on the type of message that was sent, will either create a new calendar item in their outlook, or cancel an existing item. To manage these appointments, the advisor may also use a simple command line interface to interact with the database, allowing him/her to view and cancel upcoming appointments.

- What is its structure?



- What is its Theory of Operation? (Block and flow diagrams are good here.)

An email is generated by a webform accessed by a student who wants to schedule an advising appointment. This email is sent to an advisors .engr account and rerouted by the procmail filter to an email parser. The email

parser takes the data, sends it to the database, creates a calendar attachment, which it then sends with a reformatted email to the advisor's target address. The advisor is able to view these appointments that exist in the database using the CLI, allowing them to view meetings and cancel them if necessary. Upon cancellation, an email's status will be modified to cancelled in the database, and a cancellation email will consequently be sent to the advisor, removing the item from the calendar.

- How does one install your software?

  A professor installs the program by adding the procmail file to their home directory and making alterations to the path where the email parser file is stored, as well as altering the email address to be their own. The email parser will then also need to be added at the destination directory of that path, and the user will then need to enter their email username and password, as well as a non engr account (their onid account) for the email to route to and add the calendar item. Finally, they must also update their database credentials with proper login information in both the email parser and the CLI, which can be run from any location, in order to establish a connection. Because the procmail file is responsible for piping relevant emails to the email parser, which then creates appointments which can be viewed in the curses program or in Outlook, there is no other installation required. The students don't install the software; they interact with the webform to send an appointment request to their professor.
- How does one run it?

  The procmail and filter essentially run the program for you. A student will enter an appointment request from the web form which will get caught by the procmail filter, and the program will then handle the email. You must follow the installation steps above and then you can either manage the calendar items populated by the email parser directly within outlook or with the CLI. To run the CLI, you must enter python cli.py in order to launch the interface.

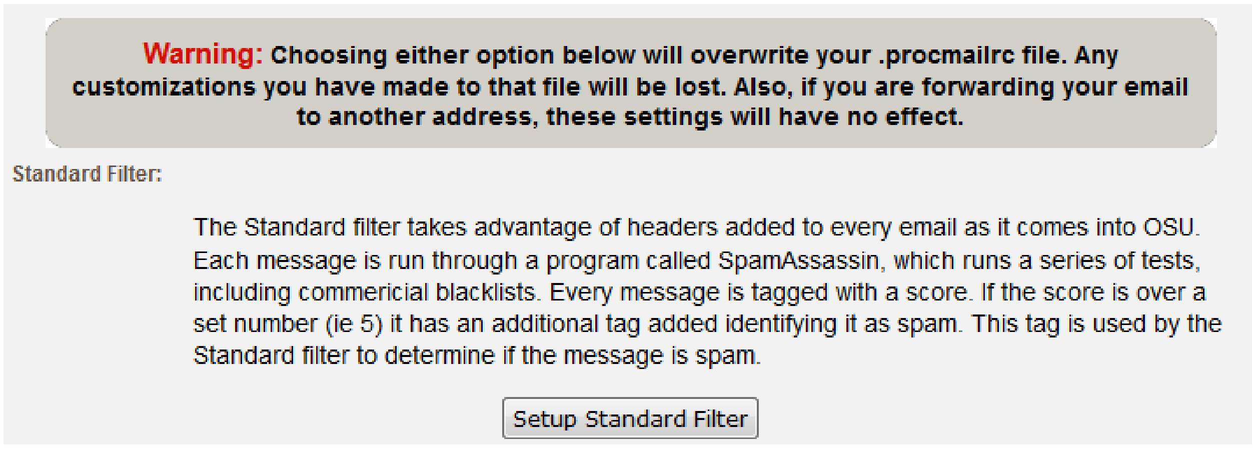
From there, you may delete appointments currently listed in the database using keyboard input.

Are there any special hardware, OS, or runtime requirements to run your software?
A Linux OS is helpful for using the command line client. There aren't any other special requirements to use the software, other than of course basic requirements such as having an engr and onid account, as well as internet access.

- Please provide a written getting started guide. This should include getting set up, as well as several examples, with screenshots, of how to use your project. If you prefer to do this in video form, you are welcome to do so in addition to the written version.

Getting Started:

<u>Installing procmail</u>:

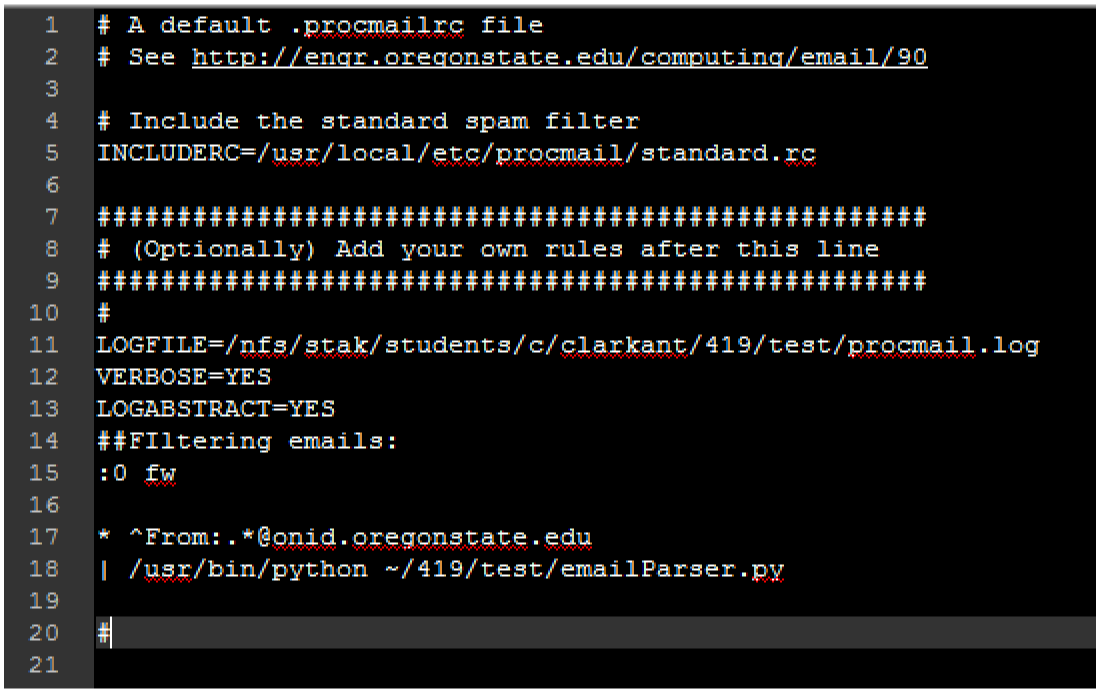


Warning: Choosing either option below will overwrite your .procmailrc file. Any customizations you have made to that file will be lost. Also, if you are forwarding your email to another address, these settings will have no effect.

Standard Filter:

The Standard filter takes advantage of headers added to every email as it comes into OSU. Each message is run through a program called SpamAssassin, which runs a series of tests, including commericial blacklists. Every message is tagged with a score. If the score is over a set number (ie 5) it has an additional tag added identifying it as spam. This tag is used by the Standard filter to determine if the message is spam.

In order to alter engr's procmail filter, go to teach (https://secure.engr.oregonstate.edu:8000/teach.php?type=procmail) and within the spam filter page, under email tools, click set up standard filter. As is always good advice in life, ignore the warning. This will cause a .procmailrc file to become available for editing in your main directory. Navigate to that file and open it.

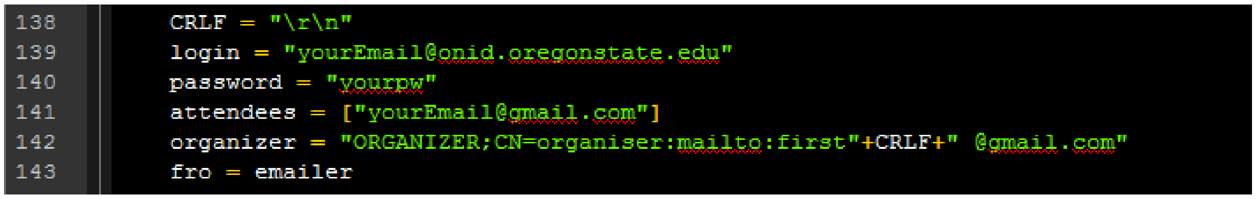


```
 1  # A default .procmailrc file
 2  # See http://engr.oregonstate.edu/computing/email/90
 3
 4  # Include the standard spam filter
 5  INCLUDERC=/usr/local/etc/procmail/standard.rc
 6
 7  ####################################################
 8  # (Optionally) Add your own rules after this line
 9  ####################################################
10  #
11  LOGFILE=/nfs/stak/students/c/clarkant/419/test/procmail.log
12  VERBOSE=YES
13  LOGABSTRACT=YES
14  ##FIltering emails:
15  :0 fw
16
17  * ^From:.*@onid.oregonstate.edu
18  | /usr/bin/python ~/419/test/emailParser.py
19
20  #
21
```

Within the file, make the following alterations. To lines 12-17. The LOGFILE is optional to check for errors and messages sent by the filter. If this is desired, alter the path to the desired directory where you will store your procmail.log file. As we can see the procmail filter will forward all emails sent from an @onid.oregonstate.edu address into the emailParser.py file. Alter this path to fit your own directory structure with the emailParser.py file being the destination.

<u>Setting up the emailParser.py file</u>

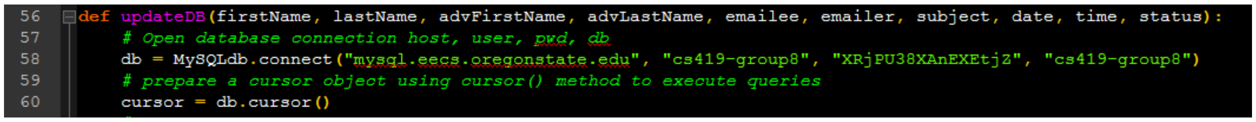
Download and place the emailParser.py file in the directory specified by your procmail filter in the step above. Next, open to file and perform the following alterations:



```
138      CRLF = "\r\n"
139      login = "yourEmail@onid.oregonstate.edu"
140      password = "yourpw"
141      attendees = ["yourEmail@gmail.com"]
142      organizer = "ORGANIZER;CN=organiser:mailto:first"+CRLF+" @gmail.com"
143      fro = emailer
```

Within the file, scroll down to lines 139-141 within the sendEmail function definition. Here, on line 139, enter the email that you will be use to send your email from, as well as its corresponding password on line 140. Then on line 141, include the email to which the message and calendar attachment will be

sent. Once this is complete, navigate back up to the updateDB function definition and enter your mySQL database login information on line 58.

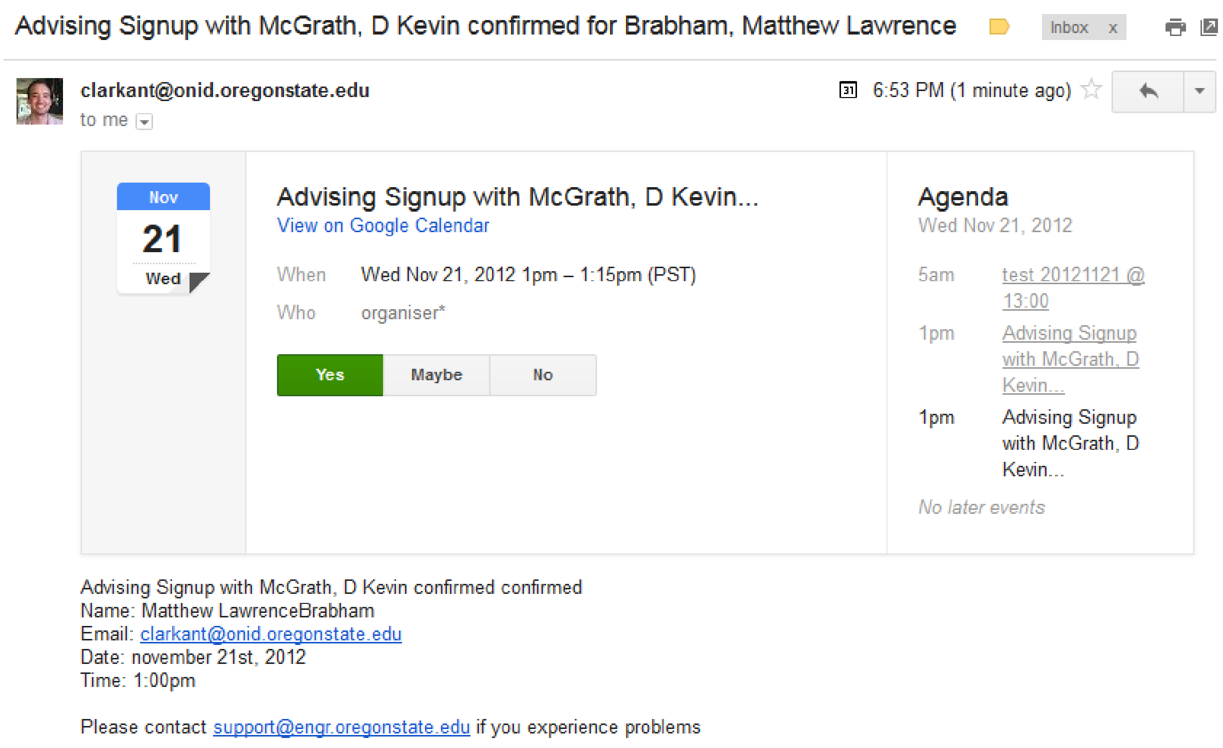


```
56 def updateDB(firstName, lastName, advFirstName, advLastName, emailee, emailer, subject, date, time, status):
57     # Open database connection host, user, pwd, db
58     db = MySQLdb.connect("mysql.eecs.oregonstate.edu", "cs419-group8", "XRjPU38XAnEXEtjZ", "cs419-group8")
59     # prepare a cursor object using cursor() method to execute queries
60     cursor = db.cursor()
```

Your emailParser.py file is now configured and can be closed.

Emails that are received will now appear in the following format format:

- **Confirmations: Will add an appointment to the calendar**



Advising Signup with McGrath, D Kevin confirmed for Brabham, Matthew Lawrence

clarkant@onid.oregonstate.edu
to me

6:53 PM (1 minute ago)

Nov 21 Wed

Advising Signup with McGrath, D Kevin...
View on Google Calendar

When Wed Nov 21, 2012 1pm – 1:15pm (PST)
Who organiser*

Agenda
Wed Nov 21, 2012

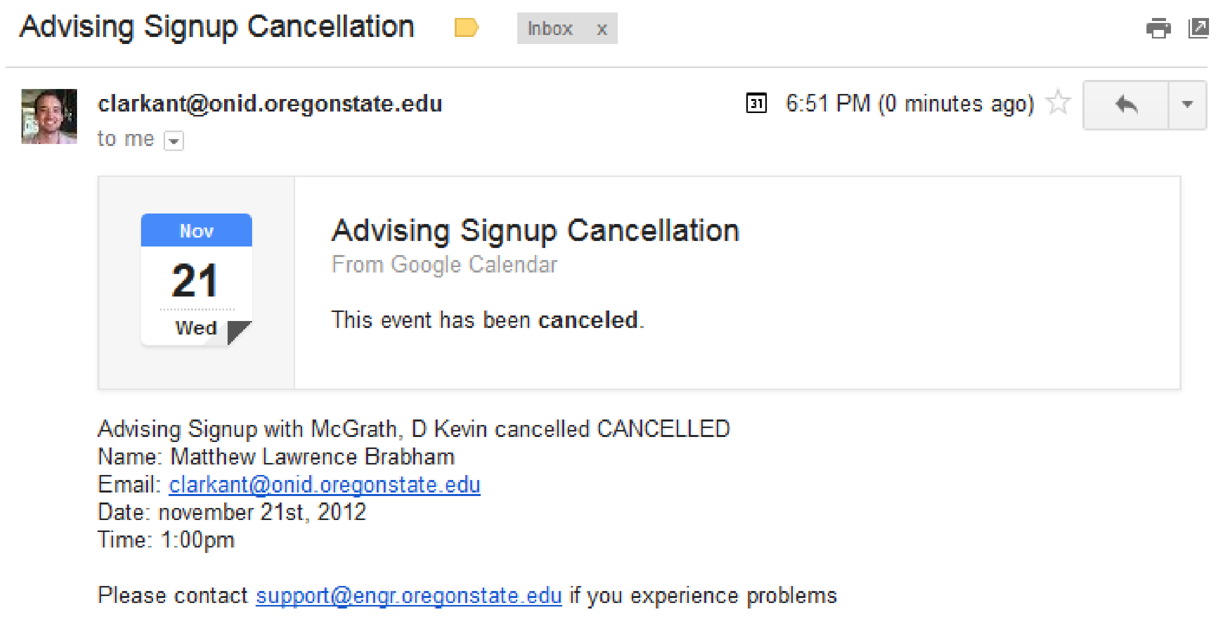
5am	test 20121121 @ 13:00
1pm	Advising Signup with McGrath, D Kevin...
1pm	Advising Signup with McGrath, D Kevin...

No later events

Advising Signup with McGrath, D Kevin confirmed confirmed
Name: Matthew LawrenceBrabham
Email: clarkant@onid.oregonstate.edu
Date: november 21st, 2012
Time: 1:00pm

Please contact support@engr.oregonstate.edu if you experience problems

- **Cancellations: Will remove an appointment on the calendar**



Advising Signup Cancellation Inbox x

clarkant@onid.oregonstate.edu
to me

6:51 PM (0 minutes ago)

Nov 21 Wed

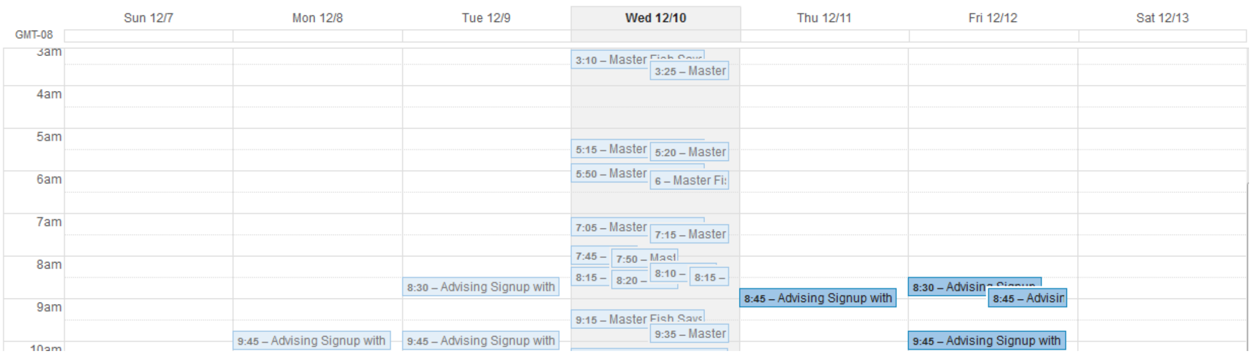
Advising Signup Cancellation
From Google Calendar

This event has been **canceled**.

Advising Signup with McGrath, D Kevin cancelled CANCELLED
Name: Matthew Lawrence Brabham
Email: clarkant@onid.oregonstate.edu
Date: november 21st, 2012
Time: 1:00pm

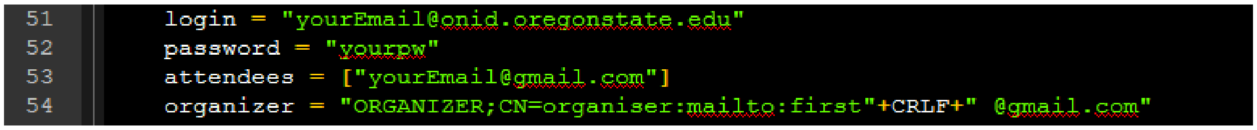
Please contact support@engr.oregonstate.edu if you experience problems

- **Calendar view: Items added to the calendar (lots of tests seen here..)**



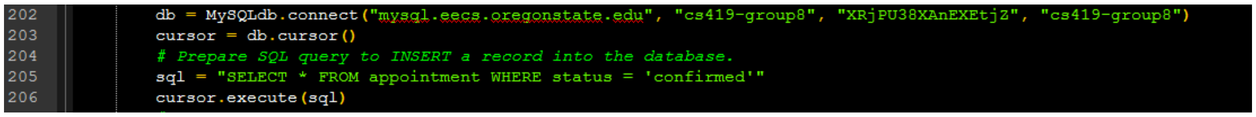
<u>**Setting up the CLI**</u>

Return to the directory where you placed the CLI.py file back in the installation process. Open the file and scroll to the lines in the sendEmail function that handle the email account. Just like in the emailParser.py file, we're going to add our email login credentials.



```
        login = "yourEmail@onid.oregonstate.edu"
        password = "yourpw"
        attendees = ["yourEmail@gmail.com"]
        organizer = "ORGANIZER;CN=organiser:mailto:first"+CRLF+" @gmail.com"
```

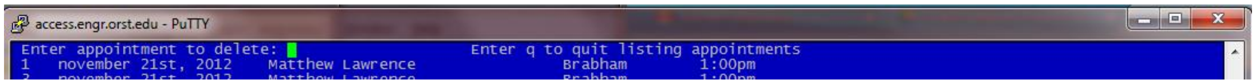
Next, go further down the file to the line where we're setting our database login, and modify the data to be reflective of the information needed to access your mySQL database.



```
        db = MySQLdb.connect("mysql.eecs.oregonstate.edu", "cs419-group8", "XRjPU38XAnEXEtjZ", "cs419-group8")
        cursor = db.cursor()
        # Prepare SQL query to INSERT a record into the database.
        sql = "SELECT * FROM appointment WHERE status = 'confirmed'"
        cursor.execute(sql)
```

The data in each of these will be the same information that you used within the emailParser program.

Once the CLI has been configured to run from your account, you will be given a display page that you can easily navigate to and will be presented with the items in your database, as well as the option to delete any given appointment.



access.engr.orst.edu - PuTTY

```
Enter appointment to delete:                            Enter q to quit listing appointments
1    november 21st, 2012     Matthew Lawrence                   Brabham         1:00pm
```

Running a delete on a given item will change its status in the database from confirmed to cancelled, removing it from the list of appointments, and sending out a cancellation email, which will be in the same format as the ones pictured above.

4. How did you learn new technology?
   - What web sites were helpful? (Listed in order of helpfulness.)

     Ryan:

     - https://docs.python.org/2/library/curses.panel.html#module-curses.panel
     - https://docs.python.org/2/library/curses.html
     - http://stackoverflow.com/questions/14200721/how-to-create-a-menu-and-submenus-in-python-curses?lq=1
     - http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/

     Well, at first, when I tried to create a menu, I did it without panels. That was a good few weeks wasted when I realized that when I had to get things connected and have it shown on separate screens, it would be way more work than what I intended to do. That is when I learned about panels and found the stack overflow site. With the previous work I did in python on the CLI side, I just converted things to panels and it made my life and my teammate's life much easier. From there, it was just plug and test from the python documentation. This project made me realize how I should appreciate documentation more than I have in the past. The stackoverflow site was what set me in the right direction.

Anthony:

Aside from the useful websites listed below, a lot of the learning came in the form of experimenting. The usage of dates was especially challenging for the calendar item. For as much as I love python, this was the first time I've really felt their syntax and formatting was unintuitive. Because I had no experience with curses or calendar items, these were especially trial and error driven. Other portions, other than the procmail filter which has decent documentation, were either easily understood through documentation or things for which I already had some form of knowledge base.

Useful websites:

- Stackoverflow.com
- python.org/doc/
- tutorialspoint.com
- pythonforbeginners.com/basics/python-datetime-time-examples
- piazza
- userpages.umbc.edu/~ian/procmail.html

Jennifer:

- http://www.procmail.org/
- http://userpages.umbc.edu/~ian/procmail.html
- https://www.freebsd.org/doc/handbook/mail-procmail.html

○ What, if any, reference books really helped?

  There were no reference books used.

○ Were there any people on campus that were really helpful?

  No people on campus were used due to the class being online. However, the information about using procmail with Oregon State email was helpful. This webpage is at http://engineering.oregonstate.edu/computing/email/90. In addition, guidance and suggestions from other students on the class' piazza page was also useful.

5. What did you learn from all this?

   We learned more skills with python than we had before, including many related to parsing, email libraries and attachments, how to use curses to make a cli, and lessons on how to improve our ability to effectively work in groups going forward.

6. One per team member.

   Anthony:

   What technical information did you learn?

   - **Python date objects**
   - **How to send emails using the smtp lib**
   - **How to make email attachments without crying too much**
   - **How to use curses to make a cli**

   What non-technical information did you learn?

   - **How to coordinate workflow and projects with other group members**
   - **How important keeping to a schedule is**
   - **How valuable documentation and tutorials can be**

   What have you learned about project work?

   - **That it is much more successful when you can interact with some one face to face. Having had many group projects over the past, this was one of the more difficult ones due to the amount of work involved, conflicting schedules, and the inability to just sit down and discuss on a whim, without a scheduled time.**

   What have you learned about project management?

   - **I've learned that project management is most successful when things are in writing and strictly adhered to.**

   What have you learned about working in teams?

   - **On a planning level, I learned that documentation is very important, as is communication as to where you currently are with your given task, so that your group members can have a better idea of where the work is at and no one falls behind.**

   If you could do it all over, what would you do differently?

   - **If I could do it all over, I would have implemented a better check system to see that we all finished our work in a timely fashion, and in line with the**

milestones document that we created. This would have eased stress and allowed us more breathing room in the closing weeks.

Ryan:

What technical information did you learn?

- **Everything about CLI**
- **Object Oriented Programming with Python**
- **Python in general**

What non-technical information did you learn?

- **How to get work done on a deadline**
- **How to recover from miscommunication mistakes**
- **The importance of documentation**

What have you learned about project work?

- **That it is never fair and the work can never be divided equally. Someone always does less work and someone more work. It's just best if you can find reliable people to have everyone put 110%, including yourself.**
- **Initiative and being on top of things are valuable traits in a programmer**
- **At the end of the day, it falls on you to get things done**

What have you learned about project management?

- **Scheduling is everything**
- **Putting time aside in the day where everyone is focused and on the same page will increase results later on.**

What have you learned about working in teams?

- **Never assume someone is on the same page as you.**
- **Have some sort of accountability parameters that members have to meet so work doesn't fall behind**

If you could do it all over, what would you do differently?

- **Don't assume your code is going to 'work' because you put time into it when trying to merge other's work into your own**
- **Be more involved with your teammates and their code earlier on**

Jennifer:

What technical information did you learn?

I learned about Procmail filtering.

I learned about how the parts of email can be piped into other programs.

What non-technical information did you learn?

- **I was reminded again how much better it is to work on a project slowly and break it up into parts, rather than trying to do it all at once.**

What have you learned about project work?

- **I learned that projects are hard to organize when a group is spread out and not all at one place.**
- **I learned that a repository is probably necessary for group work with a group that is not in the same place, so that each group member can see current progress.**

What have you learned about project management?

- **I learned that projects need to be saved in repositories so that a group can stay organized about what is done so far.**
- **I learned that projects need small steps created so that progress can be made gradually, rather than large portions of the project being completed all at once.**

What have you learned about working in teams?

- **Communication with a group is important and if a concept is not understood, it needs to be communicated with the group to see if someone else has more expertise.**
- **Its more difficult if the group is not in the same place and if the group is spread out, the project will probably take more time.**

If you could do it all over, what would you do differently?

- **I would set up a repository for our code so that group members could see the progress of other members.**
- **I would make slower steady progress throughout the quarter, rather than waiting until a deadline.**

Appendix 1: Essential Code Listings.

1) **A number of parsing functions are used within our code, such as the example below. Here we read in the the email body, which is contained in a string. Searching by key words using my best friend, the smtplib, we are able to set a number of our variables exceptionally simply. This differs from within the body function where we have to iterate over the contents of the entire body and search by key word. These will all be stored for later use in our email and database.**

```python
def parseHeader(filename):
        headers = Parser().parsestr(filename)
        #print emailee
        emailee =  headers['to']
        #print emailer
        emailer = headers['from']
        subject = headers['subject']
        #print subject
        subjectLine = headers['subject']
        headSplit = subjectLine.split()
        #Get the appointment requesters last name from the subject line and remove the ,
        lastName = (headSplit[8])[:-1]
        #Get the appointment requesters first name from the subject line and add item 10 for the middle name
        firstName = headSplit[9] + ' ' + headSplit[10]
        #Get the appointment advisor's last name from the subject line and remove the ,
        advLastName = (headSplit[3])[:-1]
        #Get the appointment advisor's first name from the subject line
        advFirstName = headSplit[4] + ' ' + headSplit[5]
```

2) **With the date objects being arguable the most frustrating part of this project, we have an example below, which is used to return dt to be used by our attachment. As time and date have already been saved, we pass them into the function. Time is then split on the '-' character in order to break apart the date. From there, we take the first object in the split array and save it as our time. This is then converted to date_in, our date with the start time appended, and passed through regex and finally set to the format desired using pythons strptime, a personal nemesis of this group.**

```python
def retDate(date, time):
        splitTime = time.split('-')
        startTime = splitTime[0]
        #endTime = splitTime[1]
        #print endTime
        date_in = date + ' ' + startTime
        date_in = re.sub(r"(st|nd|rd|th),", ",", date_in)
        dt = datetime.datetime.strptime(date_in, '%B %d, %Y %I:%M%p')
        return dt
```

3) Skipping from our email parser over to the CLI, we can view how our email and database interact within the client. Using our query, we select everything from the database that is currently marked as confirmed. We won't be re-cancelling any appointments obviously, so those are all left out of our display. The results are all then added to a list, where they are converted from their respective rows to variable names. The options are then printed to the screen to select an item from the list to delete, or to quit the program. The list is printed to the screen using a for loop and an enumerator to display all of our chosen information using self.window.addstr. The chosen item number, corresponding to that item's primary key, is then updated to be 'cancelled' and removed from the screen. This is first checked to make sure the list is not empty to prevent the program from crashing when it attempts to pop from an empty list. The columns relating to our chosen id are then stored in variable r and sent to our sendEmail function, which operates in the same manner as it does from our emailParser program.

```python
        sql = "SELECT * FROM appointment WHERE status = 'confirmed'"
        cursor.execute(sql)
        # Fetch all the rows in a list of lists.
        for row in cursor.fetchall():
            appt = {
                'id' : row[0],
                'date' : row[1],
                'advFirstName' : row[2],
                'advLastName' : row[3],
                'advEmail' : row[4],
                'stuFirstName' : row[5],
                'stuLastName' : row[6],
                'stuEmail' : row[7],
                'status' : row[8],
                'time' : row[9]
            }
            list.append(appt)
        search_term = None

        while search_term != 'q':
            color.border('|', '|', '-', '-', '+', '+', '+', '+')
            #break out of the while loop on q
            self.window.addstr(2, 25, 'Enter q to quit listing appointments')
            #PRINT THE RECORDS WE WANT FROM THE DB
            for index,record in enumerate(list):
                self.window.addstr(index+4, 2, str(record['id']))
                self.window.addstr(index+4, 6, str(record['date']))
                self.window.addstr(index+4, 29, str(record['stuFirstName']))
                self.window.addstr(index+4, 51, str(record['stuLastName']))
                self.window.addstr(index+4, 68, str(record['time']))
            search_term = None
            #GET ID INPUT FROM USER
            self.window.addstr(1, 25, 'Enter appointment to delete:')
            curses.curs_set(1)
            curses.echo()
            search_term = self.window.getstr(1,53,4)
            self.window.addstr(1, 54, search_term)
            self.window.clear()
            curses.doupdate()
            curses.curs_set(0)
            #CHANGE TO CANCELLED WITH ID SPECIFIED
            cursor.execute ("""UPDATE appointment SET status = 'cancelled' WHERE id=%s""", (search_term))
            #Email notification needs to be sent for cancelled appointment
            if(search_term != '\n' or search_term != 'q'):
                for index,record in enumerate(list):
                    if search_term in str(record['id']):
                        r = record
                        break
                if(r == 'stop'):
                    r = ''
                else:
                    sendEmail(r)
                    if(len(list) != 0):
                        if(search_term != 'q'):
                            list.pop(index)
```