

# Final Project Report - CS419

Jonathan Flessner and Seth Jenkins - Group 0

11 - December - 2014

## Introduction

In this project, our task was to improve the features and usability of the academic advisor scheduling system at Oregon State University. The system as it previously existed was a web form that sent a plain text email to the advisor, alerting them to a new appointment. If the advisor wants to add these appointments to a calendar, they must do so by hand. Our project improved this system in three main ways. First, we created a command line graphical user interface where the appointments can be seen and managed. Second, we implemented a mysql database where all the appointment information can be stored. Third, we wrote a script to catch the plain text emails sent by the appointment creation web form and add the appointments to the database.

Our clients are the advisors in the EECS department of Oregon State University. This is an important project as it will make it easier for advisors to manage their appointments. There should be less missed meetings as the improved system makes it easier to manage the requests. The advisors should have more time to spend with students and on research with their appointments being more automated. This should also improve the students experience in meeting with their advisors as the entire process is streamlined. The role of our clients was to answer any questions we had about the implementation as well as to discuss different usability options and help select the most optimal choice.

## Evolution of the project from design to delivery

From the original design of the project to its completed version, there were mostly small tweaks, and no large upheavals. The biggest change came in the form of a feature that had to be dropped after a team member was unable to continue work on the project without notification or warning. Due to this, we were unable to deliver on the requested feature of having Outlook receive a meeting request or cancellation before the deadline. Otherwise, the majority of changes came about during usability testing. One good example of this is what appointments to display on the home screen of the command line interface (CLI). Originally, the user was going to be able to view every appointment, past and future, in the CLI. During testing, it became obvious that there needed to be a cut off point, as it is not helpful to show appointments from days or weeks past. Through conversations with the client, we honed this down to showing only appointments from the past 12 hours. That way, you could always view your day at a glance, without having old appointments cluttering up the home screen.

### Theory of Operation

Our program has three primary parts: The CLI, the email parser, and the database. We use a procmail rule to send via pipe any email with “Advising Signup” in the title to the Python script, parseEmail.py. This script then reads the email by reading from fileinput.input() line by line. We are able to parse the relevant information we need from the lines of the email. We can also tell if we need to add or delete from the subject line of the email.

Once we have the necessary information, we either add or delete the appointment. If we need to add, then we can run a query to check if the advisor is in the database and get the advisor id (adid), if it isn't we add it. Then we will insert the appointment into the database using the firstname, lastname, student email, date/time, and adid. To delete, we again check if the advisor exists and get the adid. Then we will delete the appointment that has the matching firstname, lastname, student email, date/time, and adid.

The CLI gives a graphical depiction of the appointments to the user. On loading, the CLI determines who the user is through the environment variable on their machine. If the adid is not in the database, it is added. Otherwise, all the appointments for that given adid are shown on the main page. The graphical user interface is built through the curses module and the data is retrieved from the mysql database using the mysqlDB module. The CLI is structured with each page having its own function. There are global variables to track which page the user is on for functions like the 'back' button which returns the user to the previous page. To prevent against poor performance and crashes, the CLI will exit if the screen size is ever too small (smaller than 52x21).

### Program Requirements -

The CLI must be run on a unix/linux terminal.

The terminal must provide the user's valid username as an environment variable.

The program must be able to connect to the Oregon State engr servers.

The program is compatible with python 2.6.6. Other versions not tested.

The MySQLdb module must be installed: <http://sourceforge.net/projects/mysql-python/>

### Installing and running -

The command line interface is a simple python script. To install and run:

- 1.) Place the cli.py file in any directory you wish on your computer.
- 2.) Open a terminal that meets the programs requirements from above.
- 3.) Navigate to the directory where you placed the file.
- 4.) Run the file using the command 'python cli.py'.

To install the parseEmail.py script:

- 5.) Place the parseEmail.py file in a directory.

a.) For example, “/nfs/stak/students/j/jenkinss/cs419/parseEmail.py”

The procmail aspect requires the simple addition of this rule to your .procmailrc file:

```
:O: fw
* ^Subject:.*(Advising Signup)
| /usr/bin/python /nfs/stak/students/j/jenkinss/cs419/parseEmail.py
```

Replace “/nfs/stak/students/j/jenkinss/cs419/parseEmail.py” with the path to the parseEmail.py file on your machine. A sample .procmailrc file has been included in our installation package.

You must verify that you do not have forwarding enabled. Your .procmailrc will not run if you are forwarding emails from your ENGR to another email account.

Now when you receive an email to your ENGR email account with “Advising Signup” in the subject, it will be sent to the appropriate python script. Visit the following link if you need to learn more about how procmail works with your ENGR account:  
<http://engineering.oregonstate.edu/computing/email/90>.

#### Using the CLI -

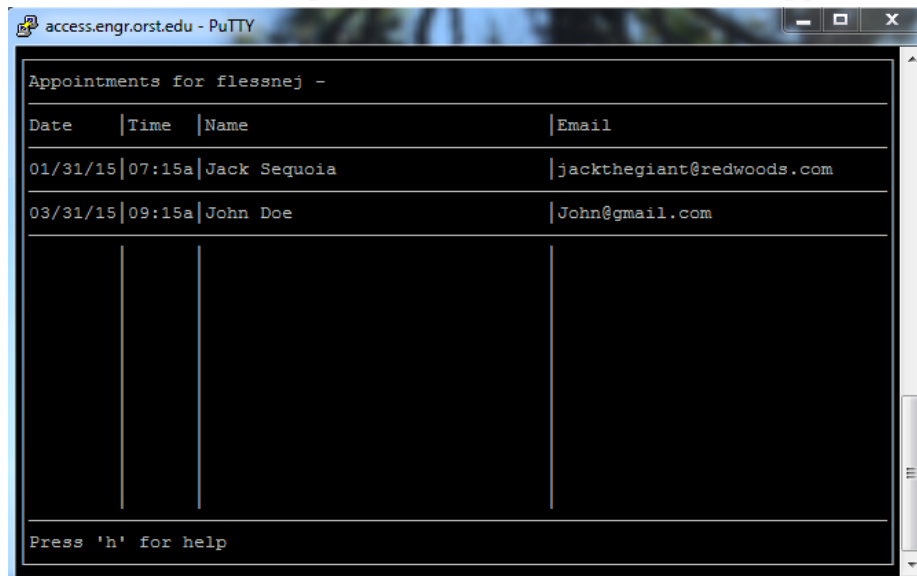
Basic controls:

The program responds to user input from single key presses.

KEY	Behavior
q	<b>Quit</b> the application
a	Go to the <b>add</b> an appointment screen
d	Go to the <b>delete</b> an appointment screen
s	Go to the <b>sort</b> appointments screen
m	Go to the <b>main</b> page
b	Go <b>back</b> to the previous page
h	Go to the <b>help</b> screen (displays this chart)

On launching the program you will arrive at the **main screen**. Here you can navigate to 4 other pages, or quit. This screen is defaulted to showing the date, time, student for your next appointment and their email. It is sorted with your next appointment showing first. Appointments more than 12 hours old will not be shown. You control the

program using single character inputs. You can press 'q' to **quit the program** at any time. Here is an example of the main screen with two appointments showing.



The screenshot shows a terminal window titled 'access.engr.orst.edu - PuTTY'. The program output is as follows:

```
Appointments for flessnej -
```

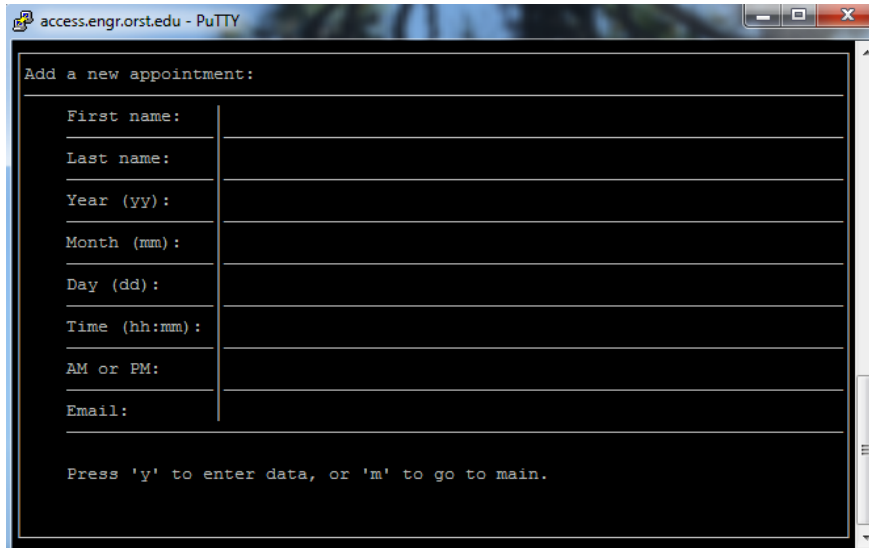
Date	Time	Name	Email
01/31/15	07:15a	Jack Sequoia	jackthegiant@redwoods.com
03/31/15	09:15a	John Doe	John@gmail.com

Press 'h' for help

On launch, the program will connect to the database where your appointments are stored. The program identifies you by the environment variable login name on your terminal. You will not be able to access your appointments by using this program unless the program can detect your login name. If this is your first time logging into the program, your username will be automatically added to the database, and you can begin using it right away. The username is found in the header of the example above.

The **help page** shows the chart from above with the basic navigation keys of the program.

At the **add page** you can add a new appointment. To start entering data you press 'y'. Before pressing 'y' you can navigate to any other page. The user experience for adding data is limited due to the curses library. You will add the relevant data by typing in the information and pressing enter to move to the next field in the form. You are unable to use tab. The data fields are: first name, last name, year, month, day, time, am or pm, and email. All fields are required. Once you have pressed enter to move to the next field, you cannot return to that field. If you make a mistake, you can enter the **cancel code: 'ccc'** into any of the fields at any point and your data will be cleared and you will return to the add page landing. See an example of the add page below. This is the landing for the add page and the user will have to press 'y' before beginning to enter the form data.



access.engr.orst.edu - PuTTY

Add a new appointment:

First name: \_\_\_\_\_

Last name: \_\_\_\_\_

Year (yy): \_\_\_\_\_

Month (mm): \_\_\_\_\_

Day (dd): \_\_\_\_\_

Time (hh:mm): \_\_\_\_\_

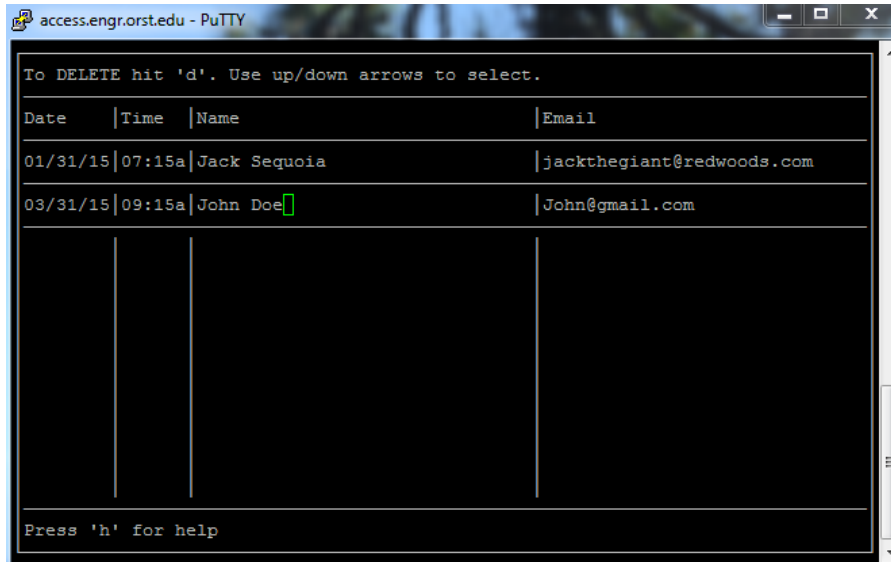
AM or PM: \_\_\_\_\_

Email: \_\_\_\_\_

Press 'y' to enter data, or 'm' to go to main.

The add page form implements client side **data validation**. In the first and last name fields, only alpha characters are allowed. If your name is hyphenated, please enter it as one word, without the hyphen or a space. Any character that is not an alpha character (a-z), including spaces or tabs, will fail. In the year field, you must enter only two digits, and they must be between the current year and 99. This program only accepts dates in the 21st century. In the month field, you must enter a valid numeric month. It must be between 1-12. Please note, this program does not allow for dates more than **1 day in the past**. All appointments older than that will not be accepted. You must enter a valid day 1-31 in the day field. The day must exist, trying to enter February 29th (on a non-leap year) or April 31st will fail. The time must be entered in 12-hour format. A colon must separate the hour from the minutes. You must enter the meridiem (AM or PM). You may also enter simply 'a' or 'p'. The email must include an characters before an @ symbol, and a dot, with more characters following each. Once you have finished adding your data, it will be automatically added to the database. You can then navigate to any other screen, or press 'a' to add another appointment.

To **delete an appointment** press 'd' to navigate to the main delete screen. The delete screen will look identical to the main screen, except for the header message and a visible cursor. Use the up and down arrow keys to move the cursor to the appointment you wish to delete. Press 'd' to select that appointment for deletion. This will move you to the **confirm deletion** page. Here you will see the appointment you have selected. To go back to the delete screen, press 'b' (or 'n'). To return to the main screen, press 'm'. To confirm that you would like to permanently delete this appointment, press 'y'. The appointment will be deleted and you will return to the main screen. An example of landing on the delete page can be seen below.



If you would like to **sort your appointments** press 's' from the main screen. You will then be taken to the sort screen. Here you have the option to sort by first name, last name, or appointment time. Once you have selected those options, you will need to select whether or not you would like the sort to be done in order (A-Z, soonest-furthest in the future) or reverse order (Z-A, furthest in the future - soonest). For example, to sort by time descending (which means furthest in the future - soonest) you would press 't' first, and then 'd' for descending. Sorting the data will keep it sorted on both the main screen and the delete screen.

Remember that with the exception of while you are entering in form data on the add page (where you would need to use the 'ccc' cancel code first) you can press 'q' to **quit** from any page.

#### Notes:

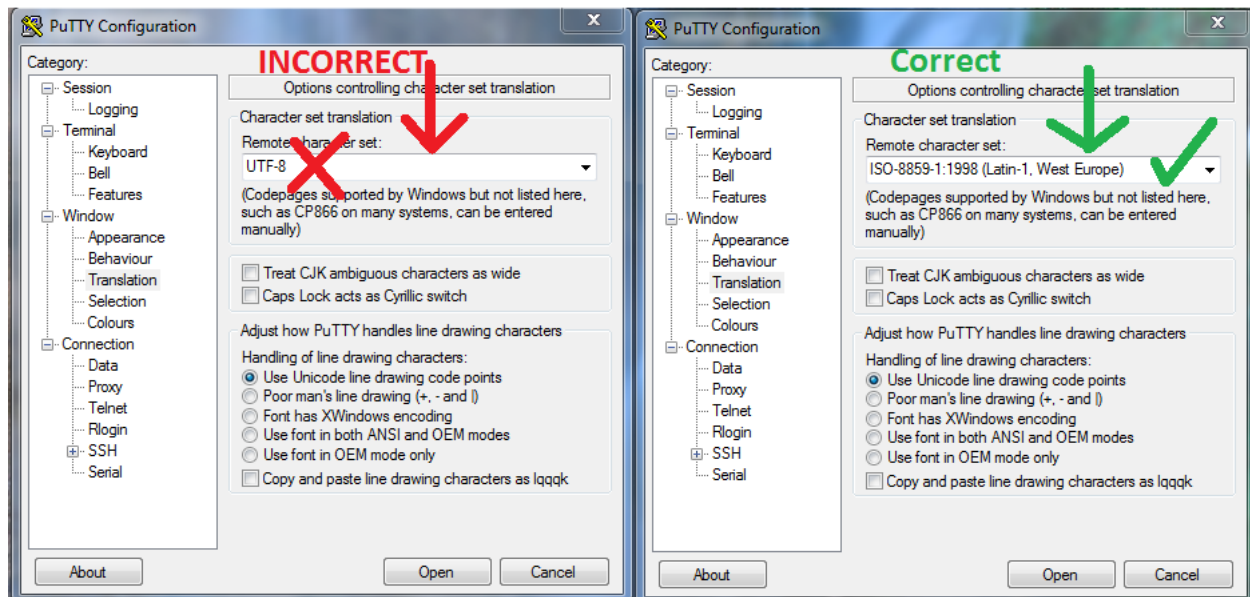
If your names or email addresses are being shown **cut off**, just increase the size of your terminal in the horizontal (x) direction.

If you would like to see **more results per page**, increase your screen size in the vertical (y) direction.

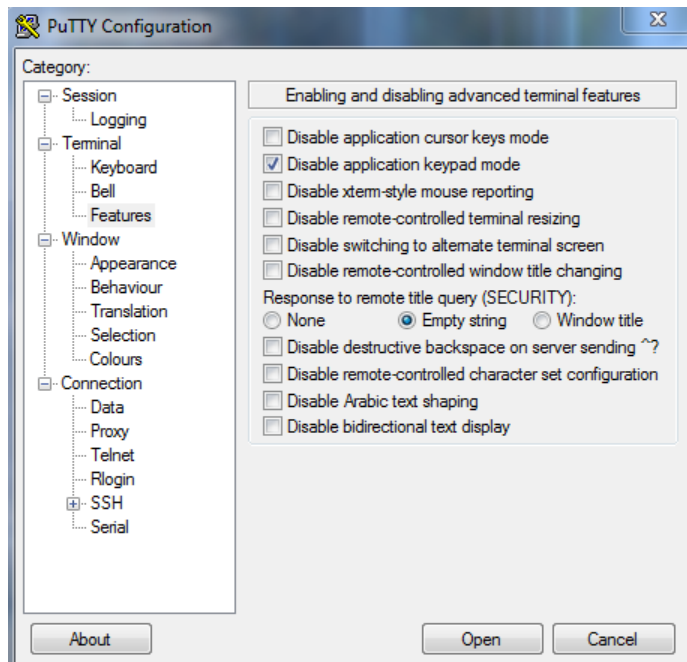
Appointments more than 120 days old will be purged for the database.

If you are getting characters as table separators (like q's, and x's) instead of clean lines, you should make sure you are using the proper character set. In your session configuration on PuTTY, make sure under the window option, and the translation sub-menu that *ISO-8859-1:1998 (Latin-1, West Europe)* is selected as your **Remote**

**character set** and not UTF-8. UTF-8 is the default for a clean download of PuTTY (as of November 2014) so this will need to be changed if that is the case.



If when entering numbers into the add field with the **numpad**, and you have numlock activated, you are getting character sequences like '^[Oq' or similar, you need to edit a setting in your terminal configuration. To do this in PuTTY, load your PuTTY session, then go to Terminal and Features. Here, make sure that "Disable application keypad mode" is checked. A screenshot with the feature correctly disabled is shown below.



### Useful sources and references

The curses library used to program the CLI was new to the programming team so a lot of research went into understanding the library. The most useful website for learning this was hands down the Python Software Foundations site on the curses library: <https://docs.python.org/2/library/curses.html>. For connecting the CLI to the mysql database we used the MySQLdb module and the documentation from its website was also invaluable: <http://www.mikusa.com/python-mysql-docs/index.html>.

There are a handful of very useful sites explaining how to set up basic procmail filters by editing your .procmailrc files. One that was especially helpful is: <http://userpages.umbc.edu/~ian/procmail.html>. There are also many useful questions on StackOverflow that were useful for more specific questions.

For the database design, our knowledge from CS 275 was instrumental. When a refresher was needed, the MySQL documentation: <https://dev.mysql.com/doc/> was a great resource.

When working on the Python script to parse the email, the experiences from CS 344 and CS 372 were important. The assignments for file I/O from CS 344 were crucial for getting information from a file/stdin. The Python documentation was also important as always: <https://docs.python.org/2/>.

### What did we learn?

The biggest takeaways for our team during this project were how to plan and bring to life a project over several months. We learned a lot about the challenges of doing this without face to face communication and the crucial informal check-ins that allows for. Technically, we learned much about python its usefulness in many different applications. The union of what we learned about the technical details, the design aspects, and working with others made this a project we all took a lot away from.

### Team member specific learning outcomes

#### Jonathan Flessner:

The technical takeaways I had from this project was greatly improving my understanding of python and its general syntax. Specifically, I also learned about how to use a python script to connect to a mysql database and about the curses library. From a non-technical perspective, I learned the most about designing a product for a user and how many options and choices that gives. I completely overhauled the usability of the CLI three or four times throughout the process. I would like to thank my family and friends for being my testers. Conversations about use from people away from the project was vital in getting a finished product I am happy with.

This was a good experience learning to plan a more complicated project and work with a team throughout several months to bring a finished product to the table. This felt



more like a real-world project than any of my other projects during this degree as it was mostly hands-off with no coddling. I knew that with this project, if we fell behind, we wouldn't be able to catch up in time. In managing this, I tried to find a balance between taking the input from everyone, without having progress slowed by making each small decision by the whole group. I think this is a test faced in many situations but it was a good learning experience for me here. If I could do this project over, I would like to have a better plan for intra-group communication and sharing of progress and ideas. I don't think we did especially poorly in this area, only that there was room for improvement.

#### Seth Jenkins:

From a technical standpoint, I learned more about Python and how easy it can be to interface things that might seem unrelated when you don't know much (or anything) about them. For example, getting the email to go from through a script, and getting that information to go to a database initially seems more complicated than it was. From a non-technical standpoint, I feel that I learned how to research ways to complete a project I initially had no idea how to solve. Compared to projects in other courses, which tend to be fairly straight forward to solve since they will obviously use the skills you are currently learning about, this was a project that was pretty random and did a good job being a culmination of skills learned in the program.

Project work is not so bad. I thought I wouldn't like it since I normally wouldn't like to give up responsibility for a part of "my" code. I think I learned that it can be ok and is actually nice to not have to worry about everything yourself. I learned that when working in teams it's good to have certain, specific things for a person to be in charge of. As far as project management, we were really hands off so I think I learned that we should still be coordinated even if we have our own responsibilities.

If I could do it all over, I think I would want to use some sort of a repository. It would make sure that we can all see the code in development which would allow us to be much more involved with the separate parts of the project. It would also make it so that we wouldn't lose access to parts of the code if a group member has to leave the group.

## Appendix 1 - Essential code listings:

### *Main CLI code pattern:*

```
#initiate the curses screen
screen = curses.initscr()          #start a new screen
curses.curs_set(0)                 #doesn't work with cygwin
screen.keypad(1)                   #allow for arrow keys and escape sequences
curses.noecho()                   #don't show key presses
dims = screen.getmaxyx()           #get screen dimensions
screenSizeCheck(dims)              #make sure dimensions are big enough
#global variables for current page and results per page
page = 0                           #for when results overflow to multiple pages
rpp = (dims[0]-7)/2                #7 is headings and borders

#go to a page function (example - main, add, delete, etc...)
mainScreen(screen, dims)

#infinite loop to wait for key press
while 1:
    #draw page information using addstr etc.
    screen.refresh()
    #get key press
    key = screen.getch()
    if key == ord('[some key]'):
        backPage = '[whatever page currently on]' #this lets the 'b' key go back
        doThatFunction() #go to whatever page or function corresponds to key
    #elif for rest of key options
```

### *MySQL statements in python:*

```
#establish connection
try:
    db = MySQLdb.connect(host=myHost, user=myUser, passwd=myPass,
db=myDBname)
except MySQLdb.Error as e:
    print type(e).__name__ + ": #" + str(e.args[0])
    print e.args[1]
    print "Could not connect to MySQL database."
    exit(1)

#get cursor
try:
    cur = db.cursor() #now connected with cursor object cur
except MySQLdb.Error as e:
    print type(e).__name__ + ": #" + str(e.args[0])
```

```
print e.args[1]
print "Could not establish MySQL cursor."
exit(1)
```

```
#parse necessary information into proper mysql format
#store as variables (ex. firstName, lastName, etc...)
#execute statements (insert and delete given as examples)
```

```
insertAppointment = "INSERT INTO Appointments (firstname, lastname, student_email,
appointment_time, adid) VALUES ('%s', '%s', '%s', '%s', %d)" % (firstName, lastName,
studentEmail, mysqlDatetime, adid)
```

```
deleteAppointment = "DELETE FROM Appointments WHERE firstname='%s' AND
lastname='%s' AND student_email='%s' AND appointment_time='%s' AND adid=%d" %
(firstName, lastName, studentEmail, mysqlDatetime, adid)
```

```
#insert executed
```

```
try:
```

```
    cur.execute(insertAppointment)
    db.commit()
```

```
except:
```

```
    print "Unable to insert appointment."
    db.rollback()
```

```
#delete executed
```

```
try:
```

```
    cur.execute(deleteAppointment)
    db.commit()
```

```
except:
```

```
    print "Unable to insert appointment."
    db.rollback()
```