Improvements for next version:
We believe we gone a long way, like every thing else it's not perfect. Based on the tests, there should be a dialog box for exporting entries to indicate the process is done. I think more visual cues will also help the user. Getting the program more accessible is would be another priority. If we had more time, we think that right-click menus, double clicks, and keyboard shortcuts will definitely improve productivity. Other than that, there are some glaring bugs that need to be fixed as well.

Source: FloydCal.py

```
#!/usr/bin/env python
# -*- coding: iso-8859-1 -*-
# generated by wxGlade 0.5 on Mon Aug 06 15:38:58 2007 from C:\Documents and
Settings\Owner\My Documents\School\ICS 413\413-Project\Calendar\src\gui\prototype2.wxg
"""
File:FloydCal.py
ICS 413

Description: This is the GUI(Graphical User Interface) for the floydCal program
"""
import wx
from wxPython.wx import *
from manager import *
from datetime import date
from datetime import time


# Define ID's for event handling
ID_EXPORT = 200
ID_EXIT   = 201
ID_HOWTO  = 202
ID_ABOUT  = 203

class toDoList(wx.Frame):
    def __init__(self, *args, **kwds):
        """The Intialization of the window frame and it's widgets and window events"""
        # begin wxGlade: toDoList.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)

        # Menu Bar
        self.toDoList_menubar = wx.MenuBar()
        self.SetMenuBar(self.toDoList_menubar)
        self.CreateStatusBar()
        menuBar = wx.Menu()
        menuBar.Append(ID_EXPORT, "Export", "Save file", wx.ITEM_NORMAL)
        menuBar.Append(ID_EXIT, "Exit", "Toodles (^_^)y", wx.ITEM_NORMAL)
        self.toDoList_menubar.Append(menuBar, "File")
        menuBar = wx.Menu()
```

```
        menuBar.Append(ID_HOWTO, "How To", "Instructions on how to use the task list",
wx.ITEM_NORMAL)
        menuBar.Append(ID_ABOUT, "About", "Credits", wx.ITEM_NORMAL)
        self.toDoList_menubar.Append(menuBar, "Help")
        # Menu Bar end
        # -- DateBlock --- Date Selection
        self.previous = wx.Button(self, -1, "Previous")
        self.datepicker = wx.DatePickerCtrl(self, -1, style=wx.DP_DROPDOWN)
        self.next = wx.Button(self, -1, "Next")
        # -- sizer_2 -- ListCtrl Cal_ControlList - The date's tasks
        listID = wx.NewId()
        self.Cal_ControlList = wx.ListCtrl(self, listID, style=wx.LC_REPORT|
wx.LC_HRULES|wx.LC_VRULES|wx.SUNKEN_BORDER)
        # -- EntryData -- Edit(Add) Entry Panel
        #Check box to complete an event
        self.isComplete = wx.CheckBox(self, -1, "")
        #Entry box for Title of event
        self.title = wx.TextCtrl(self, -1, "")
        #location label "takes place at":
        self.at = wx.StaticText(self, -1, "     at ")
        #Entry box for the location
        self.location = wx.TextCtrl(self, -1, "")

        #--Block for determining the time of an event--
        #Label with extra spaces preceeding to buffer between text and location box:
        self.Time = wx.StaticText(self, -1, "      Time:", style=wx.ALIGN_CENTRE)
        #Hours 0-23; 24-hour clock:
        self.hour = wx.SpinCtrl(self, -1, "", min=-1, max=23)
        self.hour.SetValue(-1)
        #Minutes 0-59:
        self.min = wx.SpinCtrl(self, -1, "", min=-1, max=59)
        self.min.SetValue(-1)
        #Durations available in 15 minutes intervals:
        self.duration = wx.ComboBox(self, -1, choices=["", "15", "30", "45", "60", "75", "90",
"105", "120", "135", "150", "165", "180", "195", "210", "225", "240", "255", "270", "285",
"300"], style=wx.CB_DROPDOWN)
        #duration label (so the user knows what the box is for)
        #w/extra @ the end to make space for the submit button:
        self.inMinutes = wx.StaticText(self, -1, "minutes long   ")
        self.submit = wx.Button(self,wx.NewId(),"Submit")
        self.edit = wx.Button(self,wx.NewId(),"Edit")
        self.remove = wx.Button(self,wx.NewId(),"Remove")

        self.__set_properties()
        self.__do_layout()
        # end wxGlade
        #intialize list control columns
        self.Cal_ControlList.InsertColumn(0,"Complete?",)
        self.Cal_ControlList.InsertColumn(1,"Title")
        self.Cal_ControlList.InsertColumn(2,"Location")
```

```
        self.Cal_ControlList.InsertColumn(3,"Time")
        self.Cal_ControlList.InsertColumn(4,"Duration")
        # event - selecting (or deselecting) an entry to show on EntryData sizer on click
        self.currentItem = 0
        wx.EVT_LIST_ITEM_SELECTED(self, listID, self.onItemSelected)
        wx.EVT_LIST_ITEM_DESELECTED(self, listID, self.onItemDeselected)
        wx.EVT_LEFT_DCLICK(self.Cal_ControlList, self.onDoubleClick)
        # event - submitting an entry to the listctrl Cal_ControlList
        wx.EVT_BUTTON(self,self.submit.GetId(), self.pushSubmit)
        wx.EVT_BUTTON(self,self.edit.GetId(),self.pushEdit)
        wx.EVT_BUTTON(self,self.remove.GetId(),self.pushRemove)
        # event - changing a date
        wx.EVT_BUTTON(self,self.previous.GetId(),self.pushPrev)
        wx.EVT_BUTTON(self,self.next.GetId(),self.pushNext)
        wx.EVT_DATE_CHANGED(self, self.datepicker.GetId(),self.dateChanged)
        # event - menu events : export, exit, howto, about
        wx.EVT_MENU(self, ID_EXIT, self.exitCal)
        wx.EVT_MENU(self, ID_HOWTO, self.onHowTo)
        wx.EVT_MENU(self, ID_ABOUT, self.onAbout)
        wx.EVT_MENU(self, ID_EXPORT, self.onExport)


        ###SETUP DATA STRUCTURE###
        self.m = manager()

        #update view on listctrl at the end.("garbage collection")
        self.updateView()

    def __set_properties(self):
        """The creation of the window control's variables"""
        # begin wxGlade: toDoList.__set_properties
        self.SetTitle("Floyd")
        self.SetSize((700, 450))
        self.datepicker.SetMinSize((150, 25))
        self.isComplete.SetMinSize((30, 30))
        self.title.SetMinSize((120, 24))
        self.location.SetMinSize((120, 24))
        self.hour.SetMinSize((55, 25))
        self.min.SetMinSize((55, 25))
        self.duration.SetMinSize((60, 21))
        self.duration.SetSelection(-1)
        self.remove.Enable(False)
        self.edit.Enable(False)
        # end wxGlade

    def __do_layout(self):
        """The layout where all the window's controls are placed.
           These widgets are organized in each boxSizer created.
            """
        # begin wxGlade: toDoList.__do_layout
```

```
        listFrame = wx.BoxSizer(wx.VERTICAL)
        #Intializing Boxsizers
        entryData = wx.BoxSizer(wx.HORIZONTAL)
        sizer_2 = wx.BoxSizer(wx.HORIZONTAL)
        dateBlock = wx.BoxSizer(wx.HORIZONTAL)
        buttons = wx.BoxSizer(wx.HORIZONTAL)
        #Placing GUI objects @dateBlock - The top section
        dateBlock.Add((0, 0), 0, wx.LEFT|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|
wx.SHAPED, 60)
        dateBlock.Add(self.previous, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL, 0)
        dateBlock.Add(self.datepicker, 0, wx.LEFT|wx.RIGHT, 60)
        dateBlock.Add(self.next, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL, 0)
        dateBlock.Add((0, 0), 0, wx.LEFT|wx.RIGHT|wx.ALIGN_CENTER_VERTICAL|
wx.SHAPED, 60)
        listFrame.Add(dateBlock, 0, wx.EXPAND, 0)
        #Placing GUI objects @sizer_2 - The middle
        sizer_2.Add(self.Cal_ControlList, 1, wx.EXPAND, 0)
        listFrame.Add(sizer_2, 1, wx.EXPAND, 0)
        #Placing GUI objects @entryData - The bottom section
        entryData.Add(self.isComplete, 0, wx.EXPAND|wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL|wx.SHAPED, 0)
        entryData.Add(self.title, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL|wx.SHAPED, 0)
        entryData.Add(self.at, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL|wx.SHAPED, 0)
        entryData.Add(self.location, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL|wx.SHAPED, 0)
        entryData.Add(self.Time, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL|wx.SHAPED, 0)
        entryData.Add(self.hour, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL, 0)
        entryData.Add(self.min, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL, 0)
        entryData.Add((0, 0), 0, wx.ALL|wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL|wx.SHAPED, 25)
        entryData.Add(self.duration, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL|wx.SHAPED, 0)
        entryData.Add(self.inMinutes, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL|wx.SHAPED, 0)
        buttons.Add(self.submit, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL, 0)
        buttons.Add(self.edit, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL, 0)
        buttons.Add(self.remove, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL, 0)
        listFrame.Add(entryData, 0, wx.EXPAND, 0)
        listFrame.Add(buttons, 0, wx.ALIGN_CENTER_HORIZONTAL|
wx.ALIGN_CENTER_VERTICAL, 0)
```

```
        # Layout Setup
        self.SetSizer(listFrame)
        self.Layout()
        self.Centre()
        # end wxGlade

    def pushSubmit(self, event):
        """event that submits the entryData to Create an event on the selected day's schledule"""
        #self.Cal_ControlList.InsertStringItem(row, self.title) #TODO: replace row with variable
        #Initialize Variables
        thisComp = self.isComplete.GetValue()
        thisTitle = str(self.title.GetLineText(0))
        thisLoc = str(self.location.GetLineText(0))
        if thisLoc is '':
            thisLoc = None
        hr = self.hour.GetValue()
        mn = self.min.GetValue()
        if (hr or mn) is -1:
            thisTime = None
        else:
            thisTime = time(hr, mn)
        try:
            thisDur = int(self.duration.GetValue())
        except ValueError:
            thisDur = None
        thisDate = self.currentDate()

        print "isComplete = " + str(thisComp)
        print "Title = " + thisTitle
        print "Location = " + str(thisLoc)
        #print "hours = " + str(hr)
        #print "minutes = " + str(mn)
        if thisTime is not None: print "Time = " + thisTime.isoformat()
        print "Duration = " + str(thisDur)
        print "Date =" + thisDate.isoformat()

        #Create entry
        e = entry(thisTitle, thisDate)
        e.location = thisLoc
        e.time = thisTime
        e.duration = thisDur
        e.isComplete = thisComp

        #Add to manager
        self.m + e      #'+' is overloaded to add an entry to a mangager
        #thisEntry = self.m.addEntry(thisTitle, thisDate)
        #self.m.editEntry(thisEntry, None, thisLoc, thisTime, thisDur)
        #if thisComp is True:
        #    self.m.complete(thisEntry)
```

```
        self.updateView()

        self.clearValues()

        print "GHAAAA!"

    def pushEdit(self, event):
        """event that submits the entryData that edits a selected event on the selected day's
schledule"""
        index = self.Cal_ControlList.GetFocusedItem()
        thisComp = self.isComplete.GetValue()
        thisTitle = str(self.title.GetLineText(0))
        thisLoc = str(self.location.GetLineText(0))
        hr = self.hour.GetValue()
        mn = self.min.GetValue()
        if (hr or mn) is -1:
            thisTime = None
        else:
            thisTime = time(hr, mn)
        try:
            thisDur = int(self.duration.GetValue())
        except ValueError:
            thisDur = None
        thisDate = self.currentDate()

        for x in self.m.dateList:
            if x[0] == self.currentDate():
                print "ee",thisTitle,thisLoc
                self.m.editEntry(x[1][index], thisTitle, thisLoc, thisTime, thisDur)
                if thisComp is True:
                    self.m.complete(x[1][index])
                break

        self.submit.Enable(True)
        self.edit.Enable(False)
        self.updateView()
        self.clearValues()

    def pushRemove(self, event):
        """Removes the selected entry of the current(Selected) day"""
        index = self.Cal_ControlList.GetFocusedItem()
        for x in self.m.dateList:
            if x[0] == self.currentDate():
                self.m.removeEntry(x[1][index])
                break
        self.submit.Enable(True)
        self.edit.Enable(False)
        self.remove.Enable(False)
        self.updateView()
        self.clearValues()
```

```
        pass

    def pushPrev(self, event):
        """Goes back one day"""
        #This is way overcomplicated, needs to be looked over for synergy with datepicker
methods
        temp = self.currentDate()
        newDate = temp + datetime.timedelta(-1)
        dt = self.datepicker.GetValue()
        dt.SetMonth(newDate.month - 1)
        dt.SetDay(newDate.day)
        self.datepicker.SetValue(dt)
        self.updateView()


    def pushNext(self, event):
        """Advanced date forward 1 day"""
        #This is way overcomplicated, needs to be looked over for synergy with datepicker
methods
        temp = self.currentDate()
        newDate = temp + datetime.timedelta(1)
        dt = self.datepicker.GetValue()
        dt.SetMonth(newDate.month - 1)
        dt.SetDay(newDate.day)
        self.datepicker.SetValue(dt)
        self.updateView()

    def exitCal(self, event):
        """Quits The program"""
        print "exit"
        self.Close(True)

    def onAbout(self, event):
        """Shows the "About" dialog box"""
        dlg = wx.MessageDialog(self, "Schleduling Program for ICS 413\n"
                               "2007 Eric Fletcher & Nathan Britton\n"
                               "Woot.",
                               "About FloydCal", wx.OK | wx.ICON_INFORMATION)
        dlg.ShowModal()
        dlg.Destroy()

    def onHowTo(self, event):
        """Shows the dialog box on 'howto' operate the FloydCal program"""
        dlg = wx.MessageDialog(self,
            "Fill the data fields on the bottom with the details of\n"
            "your event. The checkbox indicates whether the event has\n"
            "been completed, and the first data field is the name\n"
            "of the event.\n"
            "Click on an entry to edit or delete it.  ", "",
            wx.OK | wx.ICON_INFORMATION)
```

```
        dlg.ShowModal()
        dlg.Destroy()

    def onExport(self, event):
        """Exports the day's events into a .ics format"""
        self.m.export(self.currentDate())

    def onItemDeselected(self, event):
        """An event when an item on the ListCtrl is deselceted:
            enables add, disables edit, disables remove"""
        self.submit.Enable(True)
        self.edit.Enable(False)
        self.remove.Enable(False)

    def onItemSelected(self, event):
        """An event when an item on the ListCtrl is deselceted:
           Grab text fields and assign them to variables
           Disables add, enables edit, ensables remove"""
        index = self.Cal_ControlList.GetFocusedItem()
        sCom = self.Cal_ControlList.GetItem(index, 0).GetText()
        sTitle = self.Cal_ControlList.GetItem(index, 1).GetText()
        sLoc = self.Cal_ControlList.GetItem(index, 2).GetText()
        sTime = self.Cal_ControlList.GetItem(index, 3).GetText()
        sDur = self.Cal_ControlList.GetItem(index, 4).GetText()
        if sTime != '':
            #Taking the first two values from the isoformat string
            sHr = int(sTime[:2])
            #Taking the 3rd and 4th values for minutes
            sMn = int(sTime[3:5])

        #Display the data in the fields available
        if sCom == "True": self.isComplete.SetValue(True)
        else: self.isComplete.SetValue(False)
        self.title.SetValue(sTitle)
        self.location.SetValue(sLoc)
        if sTime != '':
            self.hour.SetValue(sHr)
            self.min.SetValue(sMn)
        self.duration.SetValue(sDur)

        #identify the entry that is selected

        self.submit.Enable(False)
        self.edit.Enable(True)
        self.remove.Enable(True)

    def onDoubleClick(self, event):
        pass

    def dateChanged(self, event):
```

```
        """Changes the date of the current day"""
        self.updateView()

    def updateView(self):
        """Refreshes the data of the Schledule every time data is add, removed, edited, or
changed."""
        self.Cal_ControlList.DeleteAllItems()
        if len(self.m.dateList) is not 0:
            for x in self.m.dateList:
                if x[0] == self.currentDate():
                    for i in range(len(x[1])):
                        dCom = x[1][i].isComplete
                        dTitle = x[1][i].title
                        dLoc = x[1][i].location
                        if dLoc is None:
                            dLoc = ''
                        dTime = x[1][i].time
                        if dTime is not None:
                            dTime = dTime.isoformat()
                        else:
                            dTime = ''
                        dDur = str(x[1][i].duration)
                        if dDur == "None":
                            dDur = ''
                        #Now fill the list
                        self.Cal_ControlList.InsertStringItem(i, str(dCom))
                        self.Cal_ControlList.SetStringItem(i, 1, dTitle)
                        self.Cal_ControlList.SetStringItem(i, 2, dLoc)
                        self.Cal_ControlList.SetStringItem(i, 3, dTime)
                        self.Cal_ControlList.SetStringItem(i, 4, dDur)
                        #print "Printing entry :" , dCom, dTitle, dLoc, dTime, dDur

    def clearValues(self):
        """Clears Values of the EntryData"""
        self.isComplete.SetValue(False)
        self.title.SetValue('')
        self.location.SetValue('')
        self.hour.SetValue(-1)
        self.min.SetValue(-1)
        self.duration.SetValue('')

    def currentDate(self):
        """The Current Date Selected"""
        return date(self.datepicker.GetValue().GetYear(),
(self.datepicker.GetValue().GetMonth() +1),self.datepicker.GetValue().GetDay())
# end of class toDoList


if __name__ == "__main__":
    app = wx.PySimpleApp(0)
```

```
wx.InitAllImageHandlers()
toDoList = toDoList(None, -1, "")
app.SetTopWindow(toDoList)
toDoList.Show()
app.MainLoop()
```

Source: Manager.py

```
#!/usr/bin/python
"""
File: manager.py
ICS 413 Project

Description: the Basic function of a schleduing calendar.

A note on the dateList Structure:
dateList[0] is the tuple of the first day.
dateList[0][0] is the actual date value of the first day. (DateTuples)
dateList[0][1] is the event list of the first day. (DateTuples)
dateList[0][1][0] is the first event of the first day. (Entry)
dateList[1] the tuple of the second day
"""
from datetime import date
from datetime import time
import datetime
import os

class manager(object):
    """Class that manages the calendar"""
    #dateList essentially is the calendar
    #takes on the form of:
    #[[date(2007, 7, 20), [1, 2, 3]], [date(2007, 7, 21), [1, 2, 3, 4]], [date(2007, 7, 22), [1, 2]]]
    #   where the lists of numbers([1, 2, 3]) represent lists of to-do items for the corresponding
day
    dateList = []
    def __init__(self):
        print 'new manager class instantiated'

    def __add__(self, other):
        if type(other) is not entry:
            print "ERROR, cannot add anything but an entry to a manager"
        else:
            temp = self.addEntry(other.title, other.date)
            self.editEntry(temp, other.title, other.location, other.time, other.duration)
            if other.isComplete is True:
                self.complete(temp)


    def addEntry(self, title, date):                #Adds a new entry to a Calendar list
        """Adds the entry to the daylist"""
        newEntry = entry(title, date)
        if (type(title) is not str) or (type(date) is not datetime.date):
            print 'Invalid entry: must have title'
            newEntry = None
        else:
            if len(self.dateList) == 0:                    #If there is nothing in the calendar yet
```

```
                subList = [newEntry.date, [newEntry]]          #Add the first date and it's first
entry
                self.dateList.append(subList)
            else:
                for i in range(len(self.dateList)):                    #i indexes each date
                    #print self.dateList[i][0], "<-->", newEntry.date
                    if self.dateList[i][0] == newEntry.date:                #If the new entry is of an
indexed date
                        self.dateList[i][1].append(newEntry)                    #Add the entry to the end of
the list for that day
                        break
                    elif self.dateList[i][0] > newEntry.date:               #If the new entry comes before
an indexed date
                        subList = [newEntry.date, [newEntry]]               #Insert the new day and it's 1st
entry
                        self.dateList.insert(i, subList)
                        break
                    elif i == (len(self.dateList)-1):                       #If the new entry comes after all
indexed dates
                        subList = [newEntry.date, [newEntry]]               #Add the new day and it's 1st
entry to the end
                        self.dateList.append(subList)
                        break
            print 'entry', newEntry.title, 'added'
        return newEntry

    def editEntry(self, tEntry, newTitle, newLocation, newTime, newDuration):
        """edits specified entry in the list"""
        for i in range(len(self.dateList)):
            if self.dateList[i][0] == tEntry.date:
                for j in range(len(self.dateList[i][1])):
                    if self.dateList[i][1][j] == tEntry:
                        #edit entry if entry not null(none)
                        if newTitle is not None:      tEntry.title = newTitle
                        if newLocation is not None:   tEntry.location = newLocation
                        if newTime is not None:       tEntry.time = newTime
                        if newDuration is not None:   tEntry.duration = newDuration
                        print 'entry', tEntry.title, 'edited'
                        break
            elif i == (len(self.dateList)-1):
                print 'entry', tEntry.title, 'not found'
                break
        #print "First Entry: ", self.dateList[0][1][0].title, "at",  self.dateList[0][1][0].time
        self.sort(tEntry.date)
        return tEntry


    def removeEntry(self, rmEntry):
        """Removes Entry"""
        if type(rmEntry) is not entry:
```

```
            print 'ERROR, must pass an entry instance'
        else:
            for i in range(len(self.dateList)):
                if self.dateList[i][0] == rmEntry.date:
                    for j in range(len(self.dateList[i][1])):
                        if self.dateList[i][1][j] == rmEntry:
                            self.dateList[i][1].remove(rmEntry)
                            if len(self.dateList[i][1]) == 0:
                                self.dateList.remove(self.dateList[i])
                            print 'entry deleted'


    def sort(self, date):
        """Sorts The Datelist by their time"""
        if type(date) is not datetime.date:
            print 'ERROR, must pass a date to sort'
        else:
            for x in self.dateList:
                if x[0] == date:
                    if len(x[1]) is 1:
                        break
                    for y in range(1, len(x[1])):
                        for i in range(len(x[1])-y):
                            j = i + 1
                #if ((x[1][i].time is None) and (x[1][j].time is not None))
                #or ((x[1][i].location is None) and (x[1][j].location is not None))
                #or (((x[1][i].time and x[1][j].time) is not None)
                    #and (x[1][i].time > x[1][j].time)):
                            if x[1][i] > x[1][j]:
                                temp = x[1][j]
                                x[1][j] = x[1][i]
                                x[1][i] = temp
                    break
                elif x[0] < date:
                    print "ERROR, no entries exist for that date"
            print 'list re-sorted'
            #print "First Entry: ", self.dateList[0][1][0].title, "at",  self.dateList[0][1][0].time

    def complete(self, entry):
        """Tells the user if a task is completed"""
        a = 'task not found'
        for i in range(len(self.dateList)):
            if self.dateList[i][0] == entry.date:
                for j in range(len(self.dateList[i][1])):
                    if self.dateList[i][1][j] is entry:
                        self.dateList[i][1][j].isComplete = True
                        a = 'task completed'
                        break
                break
        print a
```

```
def export(self, target):
    """Function that gets the floydcal data ready to be wirtten into an *.ics file format"""
    mkTrunk = True
    print os.getcwd()
    for f in os.listdir(os.getcwd() + '/'):
        if f == 'floydcal':
            mkTrunk = False
    if mkTrunk:
        os.mkdir('floydcal')

    targetDate = None
    if type(target) is entry:
        targetDate = target.date
    elif type(target) is date:
        targetDate = target
    else: print "ERROR, wrong datatype passed"

    yearString = str(targetDate.year)
    monthString = str(targetDate.month)
    dayString = str(targetDate.day)
    mkYear = True
    mkMonth = True
    mkDay = True
    for f in os.listdir('floydcal'):
        if f == yearString:
            mkYear = False
            for g in os.listdir('floydcal/'+yearString):
                if g == monthString:
                    mkMonth = False
                    for h in os.listdir('floydcal/'+yearString+'/'+monthString):
                        if h == dayString:
                            mkDay = False
                            break
                    break
            break
    if mkYear:
        os.mkdir('floydcal/'+yearString)
    if mkMonth:
        os.mkdir('floydcal/'+yearString+'/'+monthString)
    if mkDay:
        os.mkdir('floydcal/'+yearString+'/'+monthString+'/'+dayString)


    path = os.getcwd()+'/floydcal/'+yearString+'/'+monthString+'/'+dayString

    if type(target) is entry:
        self.exportEntry(target, path)
    elif type(target) is date:
        for x in self.dateList:
```

```
            if x[0] == target:
                for y in x[1]:
                    self.exportEntry(y, path)
                break

        print 'entry saved'


    def exportEntry(self, entry, path):
        """Function that writes every entry into a *.ics file"""
        apNum = ''
        num = 0
        for i in os.listdir(path):
            if (entry.title+apNum+'.ics') == i:
                num+=1
                apNum = str(num)

        beginDateString = str(entry.date.year)+str(entry.date.month)+str(entry.date.day)
        endDateString = beginDateString
        if entry.time is not None:
            beginDateString =
beginDateString+"T"+str(entry.time.hour)+str(entry.time.minute)+'00'

        if entry.duration is not None:
            hour = entry.time.hour
            min = entry.time.minute
            deltaHrs = entry.duration/60
            deltaMin = entry.duration - (deltaHrs*60)
            hour += deltaHrs
            min += deltaMin
            endDateString = endDateString +"T"+str(hour)+str(min)+'00'


        k = file(path+'/'+entry.title+apNum+'.ics', 'w')
        k.write("BEGIN:VCALENDAR\n")
        k.write("BEGIN:VTIMEZONE\n")
        k.write("TZID:Pacific/Honolulu\n")
        k.write("BEGIN:STANDARD\n")
        k.write("TZNAME:HST\n")
        k.write("END:STANDARD\n")
        k.write("END:VTIMEZONE\n")
        k.write("BEGIN:VEVENT\n")
        k.write("DTSTART:"+beginDateString+"\n") #begin date/time
        k.write("DTEND:"+endDateString+"\n") #end
        if entry.title is not None: k.write("SUMMARY:"+entry.title+"\n")
        if entry.location is not None: k.write("LOCATION:"+entry.location+"\n")
        k.write("END:VEVENT\n")
        k.write("END:VCALENDAR")
        k.close()
```

```
class entry(object):
    "A basic to-do entry"
    title = None
    date = None
    location = None
    time = None
    isComplete = None
    duration = None
    def __init__(self, initTitle, initDate):
        self.title = initTitle
        self.date = initDate
        self.isComplete = False

    #defines the greater-than relationship between any two entries
    #   entries with no time or location (tasks) are the greatest, all tasks are equal
    #   entries with time (events) are the least, events are comparable by time
    #   entries with location and no time (errands) fall between, all errands are equal
    def __gt__(self, other):
        value = None
        if type(other) is not entry:
            print "ERROR! The second value is not an entry"
        else:
            if ((self.time is None) and (other.time is not None)) or ((self.time is None) and
(self.location is None) and (other.location is not None)) or (((self.time is not None) and
(other.time is not None)) and (self.time > other.time)):
                value = True
            else: value = False
        return value
```