```python
# Evan DePosit
# New Beginnings
# capstone
# this file contains class definitions and methods for resource, resource sched, and
the algorithm for matching students with resource events

class Resource():
    def __init__(self, name, number, start, end):
        self.name= name
        self.number= number
        self.start= start
        self.end= end
        self.eventList=[]
        self.eventSched={}
        self.eventCount=None
    def print_resource(self):
        print('Name:', self.name)
        print('Number Available:', self.number)
        print('Start:', self.start)
        print('End:', self.end)


class Resource_Sched():
    def __init__(self, classList):
        self.resourceList=[]
        self.resources={}
        self.classList=classList
        self.masterResourceEventList=[]
        self.ResourceEventNumSet=set()
        self.FreeEventNumSet=set()
        self.weekLen=None
        self.resourceUse={}
        self.allEventsByNum={}
        #how is this organized by resourceName: list of events
        self.masterEventSched={}

    def make_resources_test(self):
        resources={}
        resourceList=[]

        name='ipad'
        resources[name]=Resource(name, 30, 675, 800)
        resourceList.append(name)

        name='read to self'
        resources[name]=Resource(name, 30, 675, 800)
        resourceList.append(name)

        name='partner reading'
        resources[name]=Resource(name, 30, 675, 800)
        resourceList.append(name)

        name='writing about reading'
        resources[name]=Resource(name, 30, 695, 800)
        resourceList.append(name)

        name='listening'
        resources[name]=Resource(name, 5, 675, 800)
        resourceList.append(name)

        self.resources=resources
        self.resourceList=resourceList
    #make_resources

    def make_resources(self):
        str1= "enter the number of different types of resources used for individual
reading activities"
        resources={}
        resourceList=[]
        resourceNum =input(str1 + ': ')
        resourceNum= int(resourceNum)

        for i in range(0, resourceNum):
            print('resource ', i, ':', sep='')
            name= input('Name: ')
            name = name.lower()
            number= int(input('Number available: '))
            start= input('Starting time of availability: ')
            start= tm.time_to_min(start)
            end= input('Ending time of availability: ')
            end= tm.time_to_min(end)

            resourceList.append(name)
            resources[name]=Resource(name, number, start, end)

        self.resources=resources
        self.resourceList=resourceList


    def print_all_resources(self):
        print()
        for name in self.resourceList:
            self.resources[name].print_resource()
            print()

    def make_resource_events(self, dailyEvents, numberOfDays):
        self.weekLen=numberOfDays
        for name in self.resourceList:
            count=0
            for day in range(0, numberOfDays):
                for timeBlock in dailyEvents[day]:
                    if timeBlock[0] >= self.resources[name].start and timeBlock[1]<=
self.resources[name].end:
                        for i in range(0, self.resources[name].number):

                            #make event
                            newEvent= readingGroups.Event(day, timeBlock[0],
timeBlock[1], self.resources[name].name)
                            newEvent.num=count
                            count=count+1

                            #add event to list for resource
                            self.resources[name].eventList.append(newEvent)
                            #add event to resource's sched by day
                            if day in self.resources[name].eventSched:
                                self.resources[name].eventSched[day].append(newEvent)

                            else:
                                self.resources[name].eventSched[day]=[]
                                self.resources[name].eventSched[day].append(newEvent)

                            #add event to list for resource_sched
                            self.masterResourceEventList.append(newEvent)
                            #add event to resource_sched event sched by day
                        if day in self.masterEventSched:
                            self.masterEventSched[day].append(newEvent)
                        else:
                            self.masterEventSched[day]=[]
                            self.masterEventSched[day].append(newEvent)
            self.resources[name].eventCount=count

    def number_vertices(self, vertexList):
        count=0
        for vertex in vertexList:
            vertex.num=count
            count+=1

    def set_edges(self):
        weekLen=self.weekLen
        studentList=self.classList.studentList
        studentFreeEvents=[]
        allEvents=[]

        #KEEP make list of all free events
        for student in studentList:
            for i in range(0, weekLen):
                if i in student.freeList:
                    for freeEvent in student.freeList[i]:
                        studentFreeEvents.append(freeEvent)

        #combine lists to nubmer
        allEvents= studentFreeEvents + self.masterResourceEventList
        self.number_vertices(allEvents)

        #create dictionary of all events
        allEventsByNum={}
        for event in allEvents:
            if event.num in allEventsByNum:
                print('ERROR veretex num already in dictionary')
            else:
                allEventsByNum[event.num]=event

        studentEventNumSet=set()
        for event in studentFreeEvents:
            studentEventNumSet.add(event.num)

        resourceEventNumSet=set()
        for event in self.masterResourceEventList:
            resourceEventNumSet.add(event.num)

        self.allEventsByNum=allEventsByNum
        self.ResourceEventNumSet=resourceEventNumSet
        self.FreeEventNumSet=studentEventNumSet

    def init_resource_use(self):
        #2-d dictionary first key is student name, second key is resource name
        resourceNum={}
        for student in self.classList.studentList:
            resourceNum[student.fullName]={}
            for resourceName in self.resourceList:
                resourceNum[student.fullName][resourceName]=0
        self.resourceUse=resourceNum


    def match_events(self):
        studentFreeList= self.FreeEventNumSet
        resourceEventList=self.ResourceEventNumSet

        #copy event count of each resourced to dictionary name:eventCount
        totalEvents={}
        for resourceName in self.resourceList:
            totalEvents[resourceName]=self.resources[resourceName].eventCount

        while studentFreeList:
            #print('studentFreeList not empty')
            minWeight=999999999

            #pop off a fre event
            freeEventNum=studentFreeList.pop()
            freeEvent= self.allEventsByNum[freeEventNum]

            #get day and name from free event
            studentName=freeEvent.students.fullName
            day=freeEvent.day


            #find resourceEvent at same time with lowest weight
            #for resourceEvent in self.masterEventSched[day]:
            #pop num of resourcenumset, don't use masterEventSched.  worse case all
on same day
            for resourceEventNum in resourceEventList:
                resourceEvent=self.allEventsByNum[resourceEventNum]
                resourceName= resourceEvent.type

                #already know day is equal, need to check if in resource set
                #if changing to set, already know mate is none
                if resourceEvent.day ==  day and resourceEvent.start ==
freeEvent.start:

                    #need to make sure resource name is attache to resource event
                    ammHeld=self.resourceUse[studentName][resourceName]
                    ammCirc=self.resources[resourceName].eventCount
                    edgeWeight=((100 * ammHeld)/ ammCirc)

                    if edgeWeight < minWeight:
                        minWeight = edgeWeight
                        resourceMatch=resourceEvent
                        matchNum=resourceMatch.num

            #remove match from resourceNum list
            resourceEventList.remove(matchNum)

            #match togheter vertices, not totally necessary yet
            resourceMatch.mate= freeEvent
            freeEvent.mate= resourceMatch

            #min is found, update resourceuse table for student
            resourceName=resourceMatch.type
            self.resourceUse[studentName][resourceName]+=1

            #add event to student schedule
            student=freeEvent.students
            if day in student.eventSched:
                student.eventSched[day].append(resourceMatch)
            else:
                student.eventSched[day]=[]
                student.eventSched[day].append(resourceMatch)
```

```
import readingGroups
import timeConvert as tm
```