Assignment 1 Report

Yousam Asham ashamy1

January 23, 2020

This report outlines two different versions of *date_adt.py* and *pos_adt.py*: my implementation, and my partner's implementation. My implementation of a test driver will be used to test the partner's implementation and the results will be discussed in this report, as well as answering some questions related to why these results are the way they are. In addition, some design questions will be answered, as well as my opinion about the given design specification.

1 Testing of the Original Program

The test driver was made using a combination of test cases as well as `if`, `assert`, and `print` statements. A counter of the amount of methods that passed from each of the `DateT` and `GPosT` modules. This was a development of an earlier approach, which only consisted of `if` statements and a counter. This was changed since using `assert` statements would give an error message indicating where the `assert` statement has failed. In `GPosT` though, this approach was changed since I realized the old approach was not as effective as only using `assert` and `print` statements.

The test cases used for `DateT` were picked to ensure testing for leap years, the month of february (since it is an exception), the month of february during leap years, the ends and beginning of months. These test cases also include one normal date to make sure the non-edge cases work. The ends of the month of february is considered to be edge cases since they change with every leap year.

As for the test case selection for `GPosT`, it was hard to find edge cases other than the mixing of signs of latitude and longitude, and having them gradually get close to around 90 or -90. I also found it hard to test the `GPosT` module due to my limited background knowledge of how latitude and longitude work.

My implementation's results were as expected, the methods worked as tested, but this was because they were tested individually after their initial type-up. This was done so that troubleshooting would be easier later on during the process of testing.

Throughout testing, lots of problems were encountered such as rounding and accuracy of my methods. The rounding problem was fixed by rounding the output to four decimal places and comparing it to the correct answer that was also rounded to four decimal places. Also some problems arose with the confusion in the `arrival_date` method. The confusion arose when some cases had 1 day and 24 hours of travel. Due to these kinds of cases or scenarios, some assumptions had to be added:

- *pos_adt.py: arrival_date* —date of departure to always start at time 00:00:00, or midnight.
- *date_adt.py: __init__* —User will always be inputting a correct AD date.
- *date_adt.py: equal* —A dateT object is equal to another dateT object *if and only if* they represent the same date.
- *date_adt.py: days_between* —I assumed that there are no such thing such as negative days, and converted all differences to an absolute value before returning them.

After testing my partner's code, I found out a mistake that I have made in my own program. The getter methods for my `GPosT` module were given wrong names in my own version of `GPosT` (my mehtods were called `latitude` and `longitude`, when they were supposed to called `lat` and `long`, respectively). I fixed the my `test_driver.py` and my `GPosT` module prior to starting to test my partner's code.

2 Results of Testing Partner's Code

The first time I ran my partner's code I found out about one of *my* errors that I have explained in the previous paragraph. This mistake was then fixed and testing was allowed to proceed.

My partner's `before`, `after`, and `equal` methods did not pass. This is due to some errors made in my partner's logic in the `if` statements.

I then ran into an assertion error that did not allow the testing to go on, and so I had to fix this error. This assertion error was due to my program not converting the `move` method output back to degrees. Once I added these lines, the answers were similar to each other and allowed the testing to pass. I also made a mistake where I was converting to radians when I did not need to. This mistake was also another reason that the assertion error arose.

However, after those errors, both versions' test results were consistent with each other. They both passed the test driver.

3 Critique of Given Design Specification

What I liked about this design was the separation of the modules. This alluded the *separation of concerns* principle. It made it easier to target the methods that had to be fixed when testing failed. This separation of concerns, although is not the goal for it, made this assignment feel much more organized than other tasks or assignments I had to write before for other courses. Moreover, having more than one module helped to organize the test driver and show how both modules can be incorporated together in a method like the `arrival_date` method in the `GPosT` module.

A drawback of this design specification is how generalized it is. Many assumptions had to be made, this allowed for discrepancies between the versions. For example, in the `equal` method in the `GPosT` module, my partner's version did not compare the latitude and longitude values of the current position object to the comparable object. Instead, they went directly to calculating the distance between them (and checking whether it is less than 1). Another drawback includes the unaddressed state variables (the attributes/fields of each module). For example, some versions imported and used the `datetime` module while other did not. The way the date objects are stored in the `datetime` module is, for example, different than the way I stored them. Because of that, whenever I had to use the `datetime` module, I had to create a new object.

4 Answers to Questions

(a) Some options for the state variables for the `DateT` could be a unix timestamp. A unix timestamp is a large number that houses the number of seconds since the date of January 1st 1970. Since this would only allow for a limited amount of dates to be inputed, one could implement that a negative timestamp would be the amount of seconds before that date to achieve the range of dates before January 1st 1970. Another option for the state variables for `DateT` would be a modulo representation of the day, given the month and the year. For example, the day passed would be 132, month would be 8, and year would be 2020. This would translate to a date that is in the form (132%31/8/2020) which would evaluate to the date (08/08/2020). The month and year could also be entered as number greater than 12 or 3000.

The longitude and latitude could be represented in sexagesimal degrees format (eg: $40°20'46''$N), this would allow for a better reason for the string format to be used as the state variable in the `GPosT` module. Also, a coordinate system in 3 dimensions could be used to input the latitude and longitude.

(b) The `DateT` and `GPosT` objects are *mutable*. This is mainly due to the fact that they could be accessed from outside the module. One could make them private and when that is done, they can only view the object fields through the getter methods. The specification of this assignment did not state that the object variables are to be private (using the double underscore python convention). If this was indicated in the assignment, then the objects of both modules would be private and therefore *immutable*.

(c) The unit testing framework `pytest` would be beneficial in the future since it automatically counts how many of your test passed instead of counting them manually and printing them to the user. It also allows code execution to continue after an assert statement has failed. To sum up, it is a more automated way of testing modules in python and ensures a way of running all test cases even if an `assert` statement fails to make sure that the tester is always left with complete test results after running the test driver.

(d) Some examples of software engineering failures include *NASA's Mars Climate Orbiter* and the *EDS Child Support System*. The NASA Mars Climate Orbiter was a spacecraft that was lost in space due to an engineer that failed to make a conversion between imperial units to metric units, as a result, the spacecraft was hurtled through space and was stuck in orbit around the sun.

The EDS Child Support System created software that was extremely incomopatible and resulted in the overpayment and underpayment of millions of people's child support payments. This issue has costed the UK taxpayers over a billion US dollars to this day.

Software quality is a challenge to this day due to industry thinking that it reduces agility and the amount of thought developers are able to put into the task. It is thought to reduce agility by causing the developers to focus more on the documentation of the processes which makes them less flexible. Reduced thought causes a barrier to innovation since programs have to pass a specific quality plan which distracts from creativity and the formation of new ideas that are not able to pass easily. Another factor that demotivates software quality and testing is the funds and monetary associated with software testing. Some solutions to address the challenge of software quality could be emphasis on early testing, plan for a changeable and long-term environment, and having the attitude of developing products and not projects.

(e) The Rational Design Process is a process by which software should be designed. It starts with the documentation of requirements, followed by doocumentation of the module structures, design and documentation of the interface and its internal structures, writing the programs, and finally maintain the programs. *Faking a rational design process* refers to the documentation we would produce if we had followed an ideal process. This "faking" would happen when we have identified an ideal process but cannot follow it completely.

(f) Correctness is when a software product satisfies the requirements. Reliability is when a software product does and functions just as it was intended to do or function. Robustness is when a software product behaves reasonably in unanticipated situations. When a product satisfies unstated requirements, the program is said to be robust.

(g) *Separation of concerns* is the principle that different concerns should be isolated and considered separately. Its purpose is to reduce a complex problem into a set of simpler problem, and that way we may target each concern with a parallelization of effort. Modularity is when a complex system is divided into smaller parts called modules. Modularity allows us to enable the principle of separation of concerns so that different parts of a system are considered separately creating a more efficient way of designing a system.

E Code for date_adt.py

```
## @file date_adt.py
#  @author Yousam Asham
#  @brief This module creates date related calculations using several methods. This includes getting
     the next and previous day and calculation differences between two given days.
#  @date 09-01-2020
import datetime
## @brief An ADT for the class DateT
#  @details Some assumptions made include that the user will be inputting correct dates in order for
     the module to return some correct dates.
class DateT:
  ## @brief A constructor for the DateT class
  #  @details Initializes the attributes of the DateT objects, assuming that the user inputs
       correct/valid AD dates.
  #  @param d represents the day number eg: 0-31
  #  @param m represents the month number eg: 1-12
  #  @param y represents the yea number eg: 2020
  def __init__(self, d, m, y):
      self.d = d
      self.m = m
      self.y = y
  ## @brief A getter method for the day number.
  #  @details Assumptions: None
  #  @return The day number
  def day(self):
      return self.d
  ## @brief A getter method for the month number.
  #  @details Assumptions: None
  #  @return The month number
  def month(self):
      return self.m
  ## @brief A getter method for the year number.
  #  @details Assumptions: None
  #  @return The year
  def year(self):
      return self.y
  ## @brief A method to provide the next day.
  #  @details Assumptions: Leap years occur every 4 years.
  #  @return The day after the current object
  def next(self):
      if (self.m == 2) and (self.d == 28) and (self.y % 4 != 0):
          nextDate = DateT(1, self.m+1, self.y)
      elif ((self.m == 1) or (self.m == 3) or (self.m == 5) or (self.m == 7) or (self.m == 8) or
           (self.m == 10) or (self.m == 12)) and (self.d == 31):
          nextDate = DateT(1, self.m+1, self.y)
          if (nextDate.m == 13):
              nextDate.m = 1
              nextDate.y += 1
      elif ((self.m == 4) or (self.m == 6) or (self.m == 9) or (self.m == 11)) and (self.d == 30):
          nextDate = DateT(1, self.m+1, self.y)
      else:
          nextDate = DateT(self.d+1, self.m, self.y)
      return nextDate
  ## @brief A method to provide the previous day
  #  @details Assumptions: Leap years occur every 4 years
  #  @return The day before the current object
  def prev(self):
      if (self.d == 1):
          prevDate = DateT(self.d, self.m-1, self.y)
          if prevDate.m == 0:
              prevDate.m = 12
              prevDate.y -= 1
              prevDate.d = 31
          elif (prevDate.m == 1) or (prevDate.m == 3) or (prevDate.m == 5) or (prevDate.m == 7) or
               (prevDate.m == 8) or (prevDate.m == 10) or (prevDate.m == 12):
              prevDate.d = 31
          elif (prevDate.m == 2):
              if (prevDate.y % 4 == 0):
                  prevDate.d = 29
              else:
                  prevDate.d = 28
          else:
              prevDate.d = 30
      else:
          prevDate = DateT(self.d-1, self.m, self.y)
      return prevDate
  ## @brief A method to provide whether the day provided is before the current day
```

```
#  @details Assumptions: None
#  @param d represents the day to be compared
#  @return True if the day is before the current day, false if the day is not before the current day
def before(self, d):
    if (self.y < d.y):
        return True
    elif (self.m < d.m) and (self.y == d.y):
        return True
    elif (self.d < d.d) and (self.m == d.m):
        return True
    return False
## @brief A method to provide whether the day provided is after the current day
#  @details Assumptions: None
#  @param d represents the day to be compared
#  @return True if the day is after the current day, false if the day is not after the current day
def after(self, d):
    if (self.y > d.y):
        return True
    elif (self.m > d.m) and (self.y == d.y):
        return True
    elif (self.d > d.d) and (self.m == d.m):
        return True
    else:
        return False
## @brief A method to provide whether the day provided is equal to the current day
#  @details Assumptions: A day is equal to another day if and only if they are the same day.
#  @param d represents the day to be compared
#  @return True if the day is equal to the current day, false if the day is not equal to the current
    day
def equal(self, d):
    if (self.y == d.y) and (self.m == d.m) and (self.d == d.d):
        return True
    else:
        return False
## @brief Adds a specific number of days to a DateT objects
#  @details Assumptions: None
#  @param n represents the number of days to be added
#  @return The date after n days have been added to the DateT object
def add_days(self, n):
    date = datetime.date(self.y, self.m, self.d)
    newDate = date + datetime.timedelta(days = n)
    newerDate = DateT(newDate.day, newDate.month, newDate.year)
    return newerDate
## @brief Calculates the difference in days between two dates
#  @details Assumptions: There are no things such as negative days, and so the value returned is
    always the absolute value.
#  @param d represents a date
#  @return The difference betweent the date object and d in days
def days_between(self, d):
    currentDate = datetime.date(self.y, self.m, self.d)
    dDate = datetime.date(d.y, d.m, d.d)
    diff = currentDate - dDate
    return abs(diff.days)
## @citations https://www.programiz.com/python-programming/datetime
```

F Code for pos_adt.py

```
## @file pos_adt.py
#  @author Yousam Asham
#  @brief A Global position module that implements many functions
#  @date 11-01-2020
import math
from date_adt import DateT
global earthRadius
earthRadius = 6371
## @brief An ADT for the class GPosT
class GPosT:
    ## @brief A constructor for the GPosT class
    #  @details Initializes the attributes of the GPosT objects. Some assumptions made include that
        the latitude is between -90 and 90 degrees while the longitude is between -180 to 180.
    #  @param phi represents the latitude
    #  @param lam represents the longitude
    def __init__(self, phi, lam):
        self.phi = phi
        self.lam = lam
    ## @brief A getter method for the latitude
    #  @details Assumptions: None
    #  @return The latitude
    def lat(self):
        return self.phi
    ## @brief A getter method for the longitude
    #  @details Assumptions: None
    #  @return The longitude
    def long(self):
        return self.lam
    ## @brief Compares if a position is west of the current position
    #  @details Assumptions: A position is west of another location if the it has a smaller (or more
        negative) longitude.
    #  @param p represents position to be compared to
    #  @return True if it is west of, False if not west of
    def west_of(self, p):
        if (self.lam < p.lam):
            return True
        else:
            return False
    ## @brief Compares if a position is north of the current position
    #  @details Assumptions: A position is north of another position if it has a greater (or more
        positive) latitude.
    #  @param p represents position to be compared to
    #  @return True if it is north of, False is not north of
    def north_of(self, p):
        if (self.phi > p.phi):
            return True
        else:
            return False
    ## @brief Checks if two positions are equal
    #  @details Assumptions: None
    #  @param p represents position to be compared to
    #  @return True if they are equal (or within 1 km away from each other), False if they are not
        equal
    def equal(self, p):
        if (p.lam == self.lam) and (p.phi == self.phi):
            return True
        if (self.distance(p) < 1):
            return True
        else:
            return False
    ## @brief Moves the current GPosT object to a new position
    #  @details Assumptions: None
    #  @param b represents bearing to be moved towards
    #  @param d represents distance to be moved
    def move(self, b, d):
        phi = math.radians(self.phi)
        angularDistance = (d/earthRadius)
        b = math.radians(b)
        lam = math.radians(self.lam)
        self.phi =
            math.asin((math.sin(phi)*(math.cos(angularDistance)))+((math.cos(phi))*(math.sin(angularDistance))*(math.cos(b)
        self.lam = lam + (math.atan2((math.sin(b) * math.sin(angularDistance) *
            (math.cos(phi))),((math.cos(angularDistance))-((math.sin(phi))*(math.sin(self.phi))))))
        self.phi = math.degrees(self.phi)
        self.lam = math.degrees(self.lam)
    ## @brief Calculates distance between the current object and point p
```

```
#  @details Assumptions: Distance to be returned is in kilometers
#  @param p represents the point to calculate the distance to
#  @return The distance between the two positions
def distance(self, p):
    deltaPhi = math.radians(abs(p.phi - self.phi))
    deltaLam = math.radians(abs(p.lam - self.lam))
    phi = math.radians(self.phi)
    phi2 = math.radians(p.phi)
    a = ((math.sin(deltaPhi/2))**2) + (math.cos(phi))*(math.cos(phi2))*((math.sin(deltaLam/2))**2)
    c = 2*(math.atan2((math.sqrt(a)),(math.sqrt(1 - a))))
    d = earthRadius * c
    return d
## @brief Calculates the date of arrival for someone starting at the current position on date d
    and moving to position p at speed s
#  @details Assumptions: The starting day always starts at 00:00:00
#  @param p represents the end position
#  @param d represents the date the travel starts
#  @param s represents the speed of travel
#  @return The date of arrival (a DateT object)
def arrival_date(self, p, d, s):
    distance = self.distance(p)
    days_taken = distance/s
    date_arrival = d.add_days(int(days_taken))
    return date_arrival
```

G Code for test_driver.py

```
## @file test_driver.py
#  @author Yousam Asham
#  @brief Driver to test out date_adt.py and pos_adt.py classes
#  @date 11-01-2020
from date_adt import DateT
from pos_adt import GPosT
import math

#testing for DateT
testDate1 = DateT(11,1,2020)
testDate2 = DateT(28,2,2008)
testDate3 = DateT(1,3,2004)
testDate4 = DateT(1,3,2001)
testDate5 = DateT(1,1,2000)
testDate6 = DateT(31,12,2005)
#Flags (could also be bools)
dayMethod = 0
monthMethod = 0
yearMethod = 0
prevMethod = 0
nextMethod = 0
beforeMethod = 0
afterMethod = 0
equalMethod = 0
days_betweenMethod = 0
add_daysMethod = 0

#TESTING THE DAY() METHOD
print("Testing the day() method...")
if testDate1.day() == 11:
    if testDate2.day() == 28:
        if testDate3.day() == 1:
            if testDate4.day() == 1:
                if testDate5.day() == 1:
                    assert testDate6.day() == 31
                    dayMethod = 1
                    print("DONE")

#TESTING THE MONTH() METHOD
print("Testing the month() method...")
if testDate1.month() == 1:
    if testDate2.month() == 2:
        if testDate3.month() == 3:
            if testDate4.month() == 3:
                if testDate5.month() == 1:
                    assert testDate6.month() == 12
                    monthMethod = 1
                    print("DONE")

#TESTING THE YEAR() METHOD
print("Testing the year() method...")
if testDate1.year() == 2020:
    if testDate2.year() == 2008:
        if testDate3.year() == 2004:
            if testDate4.year() == 2001:
                if testDate5.year() == 2000:
                    assert testDate6.year() == 2005
                    yearMethod = 1
                    print("DONE")

#TESTING THE NEXT() METHOD
print("Testing the next() method...")
if testDate1.next().day() == 12 and testDate1.next().month() == 1 and testDate1.next().year() == 2020:
    if testDate2.next().day() == 29 and testDate2.next().month() == 2 and testDate2.next().year() ==
         2008:
        if testDate3.next().day() == 2 and testDate3.next().month() == 3 and testDate3.next().year()
             == 2004:
            if testDate4.next().day() == 2 and testDate4.next().month() == 3 and
                 testDate4.next().year() == 2001:
                if testDate5.next().day() == 2 and testDate5.next().month() == 1 and
                     testDate5.next().year() == 2000:
                    assert testDate6.next().day() == 1 and testDate6.next().month() == 1 and
                         testDate6.next().year() == 2006
                    nextMethod = 1
                    print("DONE")
```

```
#TESTING THE PREV() METHOD
print("Testing the prev() method...")
if testDate1.prev().day() == 10 and testDate1.prev().month() == 1 and testDate1.prev().year() == 2020:
    if testDate2.prev().day() == 27 and testDate2.prev().month() == 2 and testDate2.prev().year() ==
        2008:
        if testDate3.prev().day() == 29 and testDate3.prev().month() == 2 and testDate3.prev().year()
            == 2004:
            if testDate4.prev().day() == 28 and testDate4.prev().month() == 2 and
                testDate4.prev().year() == 2001:
                if testDate5.prev().day() == 31 and testDate5.prev().month() == 12 and
                    testDate5.prev().year() == 1999:
                    assert testDate6.prev().day() == 30 and testDate6.prev().month() == 12 and
                        testDate6.prev().year() == 2005
                    prevMethod = 1
                    print("DONE")

#TESTING THE BEFORE() METHOD
print("Testing the before() method...")
if testDate1.before(testDate2) == False:
    if testDate2.before(testDate3) == False:
        if testDate3.before(testDate4) == False:
            if testDate3.before(testDate2) == True:
                if testDate5.before(testDate6) == True:
                    assert testDate6.before(testDate1) == True
                    beforeMethod = 1
                    print("DONE")

#TESTING THE AFTER() METHOD
print("Testing the after() method...")
if testDate2.after(testDate1) == False:
    if testDate3.after(testDate2) == False:
        if testDate4.after(testDate3) == False:
            if testDate2.after(testDate3) == True:
                if testDate6.after(testDate5) == True:
                    assert testDate1.after(testDate6) == True
                    afterMethod = 1
                    print("DONE")

#TESTING THE EQUAL() METHOD
print("Testing the equal() method...")
if testDate2.equal(testDate1) == False:
    if testDate3.equal(testDate2) == False:
        if testDate4.equal(testDate3) == False:
            if testDate2.equal(testDate2) == True:
                if testDate6.equal(testDate6) == True:
                    assert testDate5.equal(testDate5) == True
                    equalMethod = 1
                    print("DONE")

#TESTING THE ADD_DAYS() METHOD
print("Testing the add_days() method...")
if testDate2.add_days(1).day() == 29:
    if testDate3.add_days(2).day() == 3:
        if testDate4.add_days(3).day() == 4:
            if testDate2.add_days(4).year() == 2008:
                if testDate6.add_days(5).month() == 1:
                    assert testDate1.add_days(6).day() == 17
                    add_daysMethod = 1
                    print("DONE")
#TESTING THE days_between() METHOD
print("Testing the days_between() method...")
if testDate2.days_between(testDate1) == 4335:
    if testDate3.days_between(testDate4) == 1096:
        if testDate4.days_between(testDate1) == 6890:
            if testDate2.days_between(testDate4) == 2555:
                if testDate6.days_between(testDate5) == 2191:
                    assert testDate1.days_between(testDate6) == 5124
                    days_betweenMethod = 1
                    print("DONE")
total = dayMethod + monthMethod + yearMethod + prevMethod + nextMethod + beforeMethod + afterMethod +
    equalMethod + days_betweenMethod + add_daysMethod
print("DateT: " + str(total) + '/10 functions passed.')
#testing for GPosT

testPos1 = GPosT(10,56)
testPos2 = GPosT(47,-88)
testPos3 = GPosT(-54, -90)
testPos4 = GPosT(-22, 2)
bearing = 48
distance = 37.5
```

```
speed = 23664
#Flags (could also be bools)
latMethod = 0
longMethod = 0
west_ofMethod = 0
north_ofMethod = 0
equalMethodGPosT = 0
moveMethod = 0
distanceMethod = 0
arrival_dateMethod = 0

print("Testing the lat() method...")
assert testPos1.lat() == 10
assert testPos3.lat() == -54
latMethod = 1
print("DONE")

print("Testing the long() method...")
assert testPos2.long() == -88
assert testPos4.long() == 2
longMethod = 1
print("DONE")

print("Testing the west_of() method...")
assert testPos1.west_of(testPos2) == False
assert testPos3.west_of(testPos1) == True
west_ofMethod = 1
print("DONE")

print("Testing the north_of() method...")
assert testPos3.north_of(testPos1) == False
assert testPos4.north_of(testPos3) == True
north_ofMethod = 1
print("DONE")

print("Testing the equal() method...")
assert testPos3.equal(testPos1) == False
assert testPos4.equal(testPos4) == True
equalMethodGPosT = 1
print("DONE")

print("Testing the move() method...")
testPos2.move(bearing, distance)
assert round(testPos2.lat(),4) == 47.2251
assert round(testPos2.long(), 4) == -87.631
testPos4.move(bearing, distance)
assert round(testPos4.lat(),4) == -21.7741
assert round(testPos4.long(),4) == 2.2699
moveMethod = 1
print("DONE")

print("Testing the distance() method...")
assert round(testPos3.distance(testPos4),4) == 8209.5470
assert round(testPos2.distance(testPos1),4) == 12706.4544
distanceMethod = 1
print("DONE")

print("Testing the arrival_date() method...")
Cairo = GPosT(30.0626, 31.2497)
Toronto = GPosT(43.651070, -79.347015)
assert Cairo.arrival_date(Toronto, testDate2, speed).day() == 28
assert Toronto.arrival_date(Cairo, testDate3, speed).day() == 1
arrival_dateMethod = 1
print("DONE")
total2 = latMethod + longMethod + west_ofMethod + north_ofMethod + equalMethodGPosT + moveMethod +
    distanceMethod + arrival_dateMethod
print("GPosT: " + str(total2) + '/8 functions passed.')
```

H Code for Partner's date_adt.py

```
## @brief An ADT ...
# @file DateT.py
#  @author Parsa Abadi
#  @brief Provides the Date ADT class for representing the date.
#  @date 18/01/2020


import datetime

from math import sin, cos, sqrt, atan2, radians, asin, degrees, pi


### @brief An ADT representing different application of the date
class DateT:
    ## @brief DateT constructor
    #  @details Initializes a DateT object with getters and setters
    #  @param d is the day of the date
    #  @param m is the month of the date
    #  @param y is the year of the date

    def __init__(self, d, m, y):
        self.d = d
        self.m = m
        self.y = y

    ## @brief Gets the d parameter of the DateT
    #  @return The day of the date
    def day(self):
        return self.d
    ## @brief Gets the m parameter of the date
    #  @return The month of the date
    def month(self):
        return self.m
    ## @brief Gets the y parameter of the date
    #  @return The year of the date
    def year(self):
        return self.y

    ## @brief Return one day after inputted date of DateT object
    #  @return The following day
    def next(self):
        newDate = datetime.datetime(self.y,self.m,self.d) + datetime.timedelta(1)
        _newDate = DateT(newDate.day,newDate.month,newDate.year)
        return _newDate
    ## @brief Return the previous date of the inputted date of DateT object
    #  @return The previous day
    def prev(self):
        newDate = datetime.datetime(self.y,self.m,self.d) + datetime.timedelta(-1)
        _newDate = DateT(newDate.day,newDate.month,newDate.year)
        return _newDate
    ## @brief Checks to see if inputted date is before DateT object
    #  @return True if DateT object is after inputted d, and false otherwise
    def before(self, d):
        if self.y < d.y:
            return True
        elif self.m < d.m:
            return True
        elif self.d < d.d:
            return True
        else:
            return False
    ## @brief Checks to see if inputted date is after DateT object
    #  @return True if DateT object is before inputted d, and false otherwise
    def after(self, d):
        if self.y > d.y:
            return True

        elif self.m > d.m:
            return True
        elif self.d > d.d:
            return True
        else:
            return False
    ## @brief Checks to see if inputted date is equal to DateT object
    #  @return True if DateT object is equal to inputted d, and false otherwise
```

```
def equal(self, d):
    if self.y == d.y:
        return True

    elif self.m == d.m:
        return True
    elif self.d == d.d:
        return True
    else:
        return False
## @brief Adds number of days,n to the DateT object
#  @return DateT day + number of days added (n)
def add_days(self, n):
    z = datetime.date(self.y,self.m,self.d) + datetime.timedelta(n)
    p = DateT(z.day,z.month,z.year)
    return p
## @brief Checks to see the number of days between inputted date and DateT object
#  @return number of days between inputted date and DateT
def days_between(self, d):
    c = datetime.date(self.y,self.m,self.d)
    p = datetime.date(d.y,d.m,d.d)
    _difference = abs(c - p)
    return _difference.days
```

I Code for Partner's pos_adt.py

```
## @brief An ADT ...
# @file DateT.py
#  @author Parsa Abadi
#  @brief Provides the Date ADT class for representing the date.
#  @date 18/01/2020
from date_adt import DateT
import datetime
from math import sin, cos, sqrt, atan2, radians, asin, degrees, pi
### @brief An ADT that implements an ADT for global position coordinates
class GPosT:
    def __init__(self, __lat, __long):
        self.__long = __long
        self.__lat = __lat

    ## @brief Getters for the fields
    #  @return the field value
    def lat(self):
        return self.__lat

    def long(self):
        return self.__long
    ## @brief Checks to see if the parameter (p) is to the west of Gpost
    #  @return True if p is to the west of GPost and false otherwise
    def west_of(self, p):
        if self.__long < p.__long:
            return True
        else:
            return False
    ## @brief Checks to see if the parameter (p) is to the north of Gpost
    #  @return True if p is to the north of GPost and false otherwise
    def north_of(self, p):
        if self.__lat > p.__lat:
            return True
        else:
            return False
    ## @brief Checks to see if the parameter (p) is equal to Gpost,that is if they're <1km apart
    #  @return True if p is equal to GPost and false otherwise
    def equal(self, p):
        _diff = self.distance(p)

        if abs(_diff) <= 1:
            print(_diff)
            return True
        else:
            return False

    ## @brief Moves the current object to the new position
    #  @return True if p is to the north of GPost and false otherwise
    def move(self, b, d):
        _Latitude = radians(self.__lat)
        _Longitude = radians(self.__long)
        _bearing = radians(b)
        R = 6371.0
        self.__lat = (asin(sin(_Latitude) * cos(d / R) + cos(_Latitude) * sin(d / R) * cos(_bearing)))
        self.__long = _Longitude + atan2(sin(_bearing) * sin(d / R) * cos(_Latitude),
                                         cos(d / R) - sin(_Latitude) * sin(self.__lat))
        self.__lat = degrees(self.__lat)
        self.__long = degrees(self.__long)


    ## @brief Moves the current object to the new position
    #  @return True if p is to the north of GPost and false otherwise
    def distance(self, p):
        R = 6371.0
        _lon = self.__long - p.__long
        _lat = self.__lat - p.__lat
        a = sin(radians(_lat) / 2) ** 2 + cos(radians(self.__lat)) * cos(radians(p.__lat)) *
            sin(radians(_lon) / 2) ** 2
        c = 2 * atan2(sqrt(a), sqrt(1 - a))
        d = R * c
        return d
    ## @brief arrival date calculates when you will get to your final destination from current
    #  @return the date you get to your destination
    def arrival_date(self,p,d,s):
        #d= v/t
        #d/v=t
```

```
_distance = self.distance(p)
print(_distance)
_time = int(_distance / s)
z = datetime.date(d.y,d.m,d.d) + datetime.timedelta(_time)
_z = DateT(z.day,z.month,z.year)
return _z
```

J Citations

- https://raygun.com/blog/costly-software-errors-history/
- https://websiteseochecker.com/blog/what-is-timestamp/
- http://www.cs.toronto.edu/ sme/CSC302/notes/20-software-quality-1.pdf
- https://testpoint.com.au/11-ways-to-improve-software-quality/