ECE 498 - Python Homework 2 Fall 2020

Goals: Writing functions, programming practice.

Do the following in a single Google Colabs file and share your results with me. Keep all your sections in the order below. Write your function prototypes exactly as given so I can easily add code to test them.

1) Write a function that computes and returns the "Julian Day Number" at noon for a given Month, Day and Year, for dates in either the Julian or Gregorian calendars. The function call will look as follows:

```
      def getJDN(month, day, year = None, julian = False):
```

When the year is "None", use the current year as follows (the "`import`" should be a the top of your code).

```
      from datetime import datetime
      year = datetime.now().year
```

When "`julian`" is false, use the Gregorian calendar, otherwise use the Julian calendar. The following algorithm should work. Use it. (Note I didn't get the one on the Wikipedia page to work).

```
If month is greater than 2, take y = year and m = month
otherwise take y = year – 1 and m = month + 12

if julian is false (Gregorian calendar):
      set A to the integer part of  y/100 and then
      set B to  2 – A + the integer part of A/4
otherwise
      set B to 0 (and A doesn't matter)

the Julian Date is then the integer part of (365.25 * y) plus
      the integer part of (30.6001 * (m+1)) plus day plus 1720995 plus B
```

For testing, this online calculator seems to get correct answers: https://keisan.casio.com/exec/system/1227779487

Also, if you have "pip installed" pandas, this code works for years from 1678 to 2261 (for checking)

```
def getJDNpandas(month, day, year = None):
      if year == None:
            year = datetime.now().year
# Create the Timestamp object
      ts = pd.Timestamp(year = year,  month = month, day = day,
                  hour = 12, second = 00, tz = 'US/Eastern')
      return ts.to_julian_date()
```

2) Write a function that does the inverse of the above (given a Julian Date Number compute the month, day and year). The prototype should look like this:

```
def JDN2date(jdn, julian = False):
```

The algorithm will be as follows:

```
set Z to the integer part of jdn and (in case a non-integer jdn is given)
set F to jdn minus Z

if julian is True:
     set A to Z
else:
     set alpha to  (Z – 1867216.25) divided by 36524.25 (integer divide)
     set A to Z plus 1 plus alpha minus alpha/4  (integer divide)
set B to A plus 1524
set C to the integer part of  ((B-122.1) divided by 365.25)
set D to the integer part of (365.25 * C)
set E to the integer part of ((B - D)/30.6001)
set day to the integer part of(B - D – integer part of (30.6001 * E) + F)
set month to E - (1 if E < 13.5 else 13)       ← express this way
set year to C - (4716 if month > 2.5 else 4715)   ← express this way
return month, day, year  as at tuple
```

3) Write a day of the week function that takes a month day and year and returns the day of the week as an integer (Sunday = 0, Monday = 1, etc.) The prototype should look like this:

```
def dow(month, day, year = None, julian = False):
```

If the `year` is `None`, use the current year, as above. "`julian`" is used as above. The day of the week is found by simply adding one to the Julian Day Number and dividing by 7 and return the remainder.

4) Write a "days between dates function" with the following prototype:

```
     def ddates(month1, day1, year1, julian1, month2, day2, year2, julian2 = 
None):
```

The only optional parameter is `julian2`. If it is `none`, then use the same calendar as `julian1`. The result is obviously the Julian Day Numbers between the two dates. (Use date2 minus date1)

5) Write a "date plus days" function with the following prototype:

```
def dateplusdays(days, month, day, year = None, julian = None):
```

The usual notes from above also apply regarding `year` and `julian.` Simply compute the Julian Day Number for the date, add the number of days, and return the associated date as a tuple (month, day, year).

6) Write a test function with the following prototype:

```
def testrange(jdn1, jdn2, julian = None):
```

This function tests your first two routines by simply cycling through the range of Julian Day Numbers, converting each to the date, then taking the date and converting back to Julian Day Numbers, and comparing the result to the original.

7) Write code that calls the above test routine with both the Julian calendar (julian set to True) and Gregorian calendar (julian set to False, or missing). Use dates from January 1, year 1 to December 31, year 3000. i.e., loop from JDN 1721426 to 2451910 for the Gregorian calendar and from 1721424 to 2451923 for the Julian calendar.

8) Write code that queries the user for a date and prints the day of the week.

9) Write code that queries the user for two dates and prints the number of days between.

10) Write code that queries the user for a date and a number of days and it prints the date that many days later (or earlier if negative).

11) Try calling your "`dow`" function with the following calls, but before doing so, guess which ones will fail. Comment out those that fail (after trying).

```
print(dow(5, 10))
print(dow(5, 10, julian = False))
print(dow(5, 10, julian = True))
print(dow(5, 10, 2020))
print(dow(5, 10, 2020, True))
print(dow(5, 10, year = 2020))
print(dow(5, day = 10, year = 2020))
print(dow(5, day = 10))
print(dow(month = 5, day = 10))
print(dow(month = 5, 10))
print(dow(day = 10, month = 5))
print(dow(day = 10, month = 5, year = 2020))
print(dow(year = 2020, month = 5, day = 10))
print(dow(year = 2020, 5, 10))
print(dow(julian = False, month = 5, day = 10))
print(dow(julian = False, year = 2020, month = 5, day = 10))
print(dow(year = 2020, julian = False, day = 10, month = 5))
print(dow(year = 2020, julian = False, 5, 10))
```