Data Engineer Coding Exercises
Date Difference Challenge

PENG, HAO WEI
eric.hw.peng@gmail.com
PHONE:
+886933222535

July 11, 2022

1 Explanations

In this section, we will use **Python** language to design system, and explain every method that we utilize in the **Date Difference Challenge** problem, and explain more detailed about each step or process in design, and further calculate number of full days elapsed between two events.

1.1 Date Input From the Console

In our system, we need to write a program that accepts date input from the console, therefore, we define two input spaces to provide an environment to user to type dates at first. Besides, the date need to follow DD/MM/YYYY format, and we also consider other potential wrong input sources, therefore, we decide to implement method as follows:

Input Dates Design:

```
# we will let user to type two dates as below, however, we will use check function to check
# whether the dates are following date format, if not, we will let user to type dates again
# (while loop), until both of dates are following date format.

first_date = input("Please Enter first date: ")
second_date = input("Please Enter second date: ")

while not check_date(first_date):
    first_date = input("Please Enter first date: ")

while not check_date(second_date):
    second_date = input("Please Enter second date: ")
```

Check Dates Function:

```
# in here, we use a check_date function to check whether input dates are following DD/MM/
                                                    YYYY format
# in addition, we also suppose some specific situations as below

def check_date(date):

    # remove space from head and tail
    date = date.strip()

    LenQ = len(date)

    # if length of DD/MM/YYYY is not equal to 10, or index 2 or index 5 not equal to
                                                        character /
    # if not follow format, we will return False
    if LenQ != 10 or date[2] != "/" or date[5] != "/":
        return False

    # use split to split day, month, year individually
    day, month, year = date.split("/")

    # check whether day, month, year are all digits (DD/MM/YYYY)
    # if not, we will return False
    if not (day.isdigit() and month.isdigit() and year.isdigit()):
        return False

    day, month, year = int(day), int(month), int(year)

    table = get_month_day_table(year)


    # in addition, we also need to check whether day, month, year are in reasonable regions
    # year (not euqal to 0)
```

```
    # month (need between 1 ~ 12)
    # day (need to make decision by month, however, in February, in our system, we will
                                                    consider leap year or not)
    # if not follow format as above, we will return False
    if year == 0 or not (1 <= month <= 12) or not (1 <= day <= table[month]):
        return False

    return True

def get_month_day_table(year):
    # check current year whether need to add one more day in February (check leap year)
    table = [False, 31, 28 + check_leap(year), 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

    return table

def check_leap(year):
    # check whether current year is leap year or not
    if year % 400 == 0 or year % 100 != 0 and year % 4 == 0:
        return True
    else:
        return False
```

1.2 Dates Calculation

On the basis of problem, we understand that when we calculate number of full days elapsed between two events, we do not need to consider the begin date and end date. In our system, we make some assumptions and design our calculation function as follows:

Count Days Function:

```
def count_day_between_two_dates(first_date, second_date):

    # remove space from head and tail
    # use split to split day, month, year individually
    f_day, f_month, f_year = first_date.strip().split("/")
    s_day, s_month, s_year = second_date.strip().split("/")

    # in our system, we suppose that first date is smaller than second date
    # if user type first date is larger than second date, we will change position
    # and do calculation again, it will not stop system operation
    if f_year + f_month + f_day > s_year + s_month + s_day:
        return count_day_between_two_dates(second_date, first_date)

    # convert string into integer type
    f_day, f_month, f_year = int(f_day), int(f_month), int(f_year)
    s_day, s_month, s_year = int(s_day), int(s_month), int(s_year)


    # in here, we recevie each day (month) of current year, and also check whether
    # it is leap year or not
    f_tb, fy = get_month_day_table(f_year), 365 + check_leap(f_year)
    s_tb, sy = get_month_day_table(s_year), 365 + check_leap(s_year)

    # in here, we will calculate total days (how many months before current month)
    # plus day of current month (begin of year to current date)
    cnt_f_day = sum(f_tb[1 : f_month]) + f_day
    cnt_s_day = sum(s_tb[1 : s_month]) + s_day

    # in first situation, if both dates with same year
    # we just need to calculate second date minus first
    # date, however, we do not need to consider begin and
    # end date, therefore, we need minus 1 additionally
    if f_year == s_year:
        return cnt_s_day - cnt_f_day - 1

    # in second situation, if both dates with different year
    # we need to calculate how many days remain from the first date
    # next, we need to consider how many years between both dates
    # in addition, each year need to check whether it is leap year
    # or not, at the last place, we need to add total days from begin
    # to current date of second date
    else:
        # remain days from the first date
        diff = fy - cnt_f_day - 1

        total_day = 0

        # how many years between both dates
        for cur_year in range(f_year + 1, s_year):
            total_day = total_day + 365 + check_leap(cur_year)

        # add total days of second date
        return diff + total_day + cnt_s_day
```

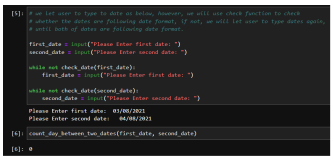
1.3 Test Results

In this section, we use test cases on the problem to check whether our system is correct, moreover, we also consider the edge cases as below:

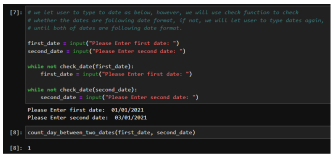
Test Cases

- 03/08/2021 − 04/08/2021 = 0 days (edge case)
- 01/01/2021 − 03/01/2021 = 1 days (edge case)
- 02/06/1983 − 22/06/1983 = 19 days
- 04/07/1984 − 25/12/1984 = 173 days
- 03/01/1989 − 03/08/1983 = 1979 days

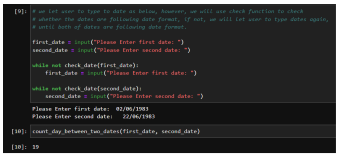
Plot Results



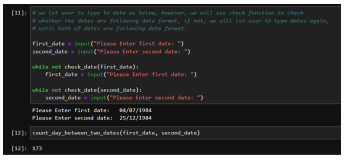
(a)



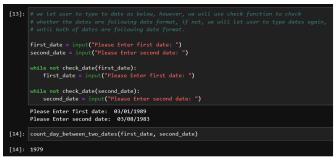
(b)



(c)



(d)



(e)

1.4 Fit into Bigger System

On my personal note, I consider that the current system design is suitable for bigger system, some reasons illustrating my perspectives will be shed lights on as follows. In the first place, we can understand that the bounded of years are 0000 to 2022, when we want to do calculation process, we only need to implement at most 2022 loop for system, for system, it is not a big value (computations). Besides, we hypothesize plenty of conditions in system to avoid system disruption, and further make system more efficiency.