EEE 111 Software Project 1 Milestone 1

Submitted a few seconds ago • Score: 25.00 Status: Accepted

✔ Test Case #0	✔ Test Case #1	✔ Test Case #2
✔ Test Case #3	✔ Test Case #4	✔ Test Case #5
✔ Test Case #6	✔ Test Case #7	✔ Test Case #8
✔ Test Case #9	✔ Test Case #10	✔ Test Case #11
✔ Test Case #12	✔ Test Case #13	

Submitted Code

Language: Python 3 Open in editor

```
#!/bin/python

# Rick's Great Plan for Success
#  1) Validate date range
#     [/] Within date limit
#     [/] All inputs are integers
#     [/] Valid dates
#  2) Number crunch
#     [/] Compute total days
#     [/] Compute weekdays
#         [/] Compute weekends
#     [/] Compute leap years
#     [ ] Compute holidays falling on weekdays
#         [ ] New Year
#         [ ] Labor Day
#         [ ] All Saint's Day
#         [ ] Christmas
#     [ ] Compute work days

# To abstract calendar math
import datetime
# To memoize commonly used functions
from functools import lru_cache
# For syntactic sugar and to make the date range immutable
from collections import namedtuple
DateRange = namedtuple("DateRange", ["start", "end"])


def get_user_input():
```

```
    """Gets date range as user input

    Raises:
        Exception: If input dates are out of date limit (January 1, 1971 to December 31, 2020)
        ValueError:
          If at least one of the following occurs:
            - At least one of the inputs is not an integer
            - At least one of the input dates is not a valid date

    Returns:
        DateRange: The start and end dates as datetime objects (contained in a namedtuple)
    """
    # Gets user input in M\nD\nYYYY format for the start date
    # Uses map to convert string input to integers and stores the values in a tuple
    start_instrings = ["Enter start month: ",
                       "Enter start day: ", "Enter start year: "]
    start_date = tuple(int(input(s)) for s in start_instrings)
    # Gets user input in M\nD\nYYYY format for the end date
    # Uses map to convert string input to integers and stores the values in a tuple
    end_instrings = ["Enter end month: ",
                     "Enter end day: ", "Enter end year: "]
    end_date = tuple(int(input(s)) for s in end_instrings)

    # Checks if each year is within the date limit
    if start_date[2] < 1971 or end_date[2] > 2020:
        raise Exception("Input date/s outside date limit.")

    # Cyclic rotation of elements (because I really really **really** want to unpack)
    # Source: https://www.geeksforgeeks.org/python-shift-last-element-to-first-position-in-list/
    start_date, end_date = start_date[-1:] + \
        start_date[:-1], end_date[-1:] + end_date[:-1]

    # As you can see unpacking makes the line smaller and more readable
    # return DateRange(datetime.date(start_date[2], start_date[0], start_date[1]),
datetime.date(end_date[2], end_date[0], end_date[1]))
    return DateRange(datetime.date(*start_date), datetime.date(*end_date))


@lru_cache(maxsize=None)
def compute_total_days(start, end):
    """Computes the total number of days between the start and end dates

    Args:
        start (datetime.date): The start date
        end (datetime.date): The end date

    Returns:
        int: The total number of days between the start and end dates
    """
    # Use the datetime module to subtract the dates (+1 if inclusive)
    return (end - start).days + 1


@lru_cache(maxsize=None)
def compute_weekends(start, end):
    """Computes the total number of weekend days between the start and end dates

    Args:
        start (datetime.date): The start date
        end (datetime.date): The end date

    Returns:
        int: The total number of weekend days between the start and end dates
    """
    # Initialize the weekends counter
    weekends = 0

    # Do-while loop (to check the start date falls on a weekend too)
```

```
    while True:
        # Check if the day falls on a weekend
        if start.weekday() == 5 or start.weekday() == 6:
            weekends += 1

        # The loop checks the days between the start date (inclusive) and
        #   the next occurence of the end date's day of the week
        if start.weekday() == end.weekday():
            break

        # Increment the start date by one day
        start += datetime.timedelta(days=1)

    # Once the start date and the end date fall on the same day of the week,
    #   we can just find the number of weeks between them and multiply
    #   by two
    weekends += ((end - start).days // 7) * 2
    return weekends


@lru_cache(maxsize=None)
def compute_weekdays(start, end):
    """Computes the total number of weekdays between the start and end dates

    Args:
        start (datetime.date): The start date
        end (datetime.date): The end date

    Returns:
        int: The total number of weekdays between the start and end dates
    """
    # Subtracts the total number of weekend days from the total number of days
    return compute_total_days(start, end) - compute_weekends(start, end)


@lru_cache(maxsize=None)
def compute_leap_years(start, end):
    """Computes the total number of leap days between the start and end dates

    Args:
        start (datetime.date): The start date
        end (datetime.date): The end date

    Returns:
        int: The total number of leap days between the start and end dates
    """
    # Generate the leap years between 1971 and 2020 inclusive
    leap_years = tuple(1972 + 4*x for x in range(13))

    # Looks for the closest leap year greater than or equal to the start year
    min_leap_year = 0
    for leap_year in leap_years:
        if leap_year >= start.year:
            min_leap_year = leap_year
            break

    # Looks for the closest leap year less than or equal to the end year
    max_leap_year = 0
    for leap_year in reversed(leap_years):
        if leap_year <= end.year:
            max_leap_year = leap_year
            break

    # Gets the number of leap years between the start and end year
    # Note that if the leap year in between is just the same year it will zero out, thus the +1
    leap_days_between = ((max_leap_year - min_leap_year) // 4) + 1

    # If the start date occurs after Feb 29th of that year, we don't consider
```

```
165     if (start - datetime.date(min_leap_year, 2, 29)).days > 0:
166         leap_days_between -= 1
167 
168     # If the end date occurs before Feb 29th of that year, we don't consider
169     if (datetime.date(max_leap_year, 2, 29) - end).days > 0:
170         leap_days_between -= 1
171 
172     return leap_days_between
173 
174 
175 @lru_cache(maxsize=None)
176 def compute_holidays(start, end):
177     return 0
178 
179 
180 @lru_cache(maxsize=None)
181 def compute_workdays(start, end):
182     return 0
183 
184 
185 if __name__ == "__main__":
186     # Getting user input and deals with errors caused by invalid output
187     try:
188         dr = get_user_input()
189     except:
190         print("\nInvalid input. Exiting Program.")
191         exit()
192 
193     # Computing the total number of days between start and end date
194     print("\ntotal days from start date to end date:",
195           compute_total_days(dr.start, dr.end))
196 
197     # Computing the total additional days from leap years
198     print("\ntotal additional days from leap years:",
199           compute_leap_years(dr.start, dr.end))
200 
201     # Computing the total number of weekend days
202     print("\ntotal weekends:",
203           compute_weekends(dr.start, dr.end))
204 
205     # Computing the total number of weekdays
206     print("\ntotal days without weekends:",
207           compute_weekdays(dr.start, dr.end))
208 
```