EEE 111 Software Project 1 Milestone 2

Submitted a few seconds ago • Score: 75.00 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	✔ Test Case #14
✔ Test Case #15	✔ Test Case #16	✔ Test Case #17
✔ Test Case #18	✔ Test Case #19	✔ Test Case #20
✔ Test Case #21	✔ Test Case #22	✔ Test Case #23
✔ Test Case #24	✔ Test Case #25	✔ Test Case #26
✔ Test Case #27	✔ Test Case #28	✔ Test Case #29
✔ Test Case #30	✔ Test Case #31	✔ Test Case #32
✔ Test Case #33	✔ Test Case #34	✔ Test Case #35
✔ Test Case #36	✔ Test Case #37	✔ Test Case #38
✔ Test Case #39	✔ Test Case #40	✔ Test Case #41
✔ Test Case #42	✔ Test Case #43	

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
```

```python
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        Exception: If input dates are out of date limit (January 1, 1971 to December 31, 2020)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
    start_instrings = ["Enter start month: ",
                       "Enter start day: ", "Enter start year: "]
    raw_start_date = tuple(input(s) for s in start_instrings)
    # Gets user input in M\nD\nYYYY format for the end date
    end_instrings = ["Enter end month: ",
                     "Enter end day: ", "Enter end year: "]
    raw_end_date = tuple(input(s) for s in end_instrings)

    # Uses map to convert string input to integers and stores the values in a tuple
    start_date = tuple(map(int, raw_start_date))
    end_date = tuple(map(int, raw_end_date))

    # Checks if each year is within the date limit
    if not(1971 <= start_date[2] <= 2020 and 1971 <= 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 if 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
    if (start - datetime.date(min_leap_year, 2, 29)).days > 0:
        leap_days_between -= 1

```

```
    # If the end date occurs before Feb 29th of that year, we don't consider
    if (datetime.date(max_leap_year, 2, 29) - end).days > 0:
        leap_days_between -= 1

    return leap_days_between


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

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

    Returns:
        dict: The number of relevant occurrences per holiday and the total number of holidays between the start and end dates
    """
    # The list of holidays and their given dates every year
    holiday_dates = {
        "new year holiday:": (1, 1),
        "labor day holiday:": (5, 1),
        "all saints day holiday:": (11, 1),
        "christmas holiday:": (12, 25)
    }

    # Initialize the count of occurrences per holiday
    holiday_counts = {holiday: 0 for holiday in holiday_dates.keys()}
    # For loop to go through each holiday
    for holiday in holiday_dates.keys():
        # Sets the year for when counting the occurrences start
        count_start = start.year
        # If the holiday occurs before the start date, we disregard it
        if (start - datetime.date(start.year, *holiday_dates[holiday])).days > 0:
            count_start += 1
        # Sets the year for when counting the occurrences end
        count_end = end.year
        # If the holiday occurs after the end date, we disregard it
        if (datetime.date(end.year, *holiday_dates[holiday]) - end).days > 0:
            count_end -= 1
        # For loop to go through each year in the counting range
        for year in range(count_start, count_end + 1):
            # If the holiday falls on a weekday, we increment the occurrence count
            if datetime.date(year, *holiday_dates[holiday]).weekday() < 5:
                holiday_counts[holiday] += 1

    # The total number of holidays is the sum of the counts of each holiday
    holiday_counts["total holidays:"] = sum(holiday_counts.values())

    # Returns the dictionary with complete counts
    return holiday_counts


@lru_cache(maxsize=None)
def compute_workdays(start, end):
    """Computes the total number of workdays 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 workdays between the start and end dates
    """
    # Subtracts the total number of holidays from the total number of weekdays
    return compute_weekdays(start, end) - compute_holidays(start, end)["total holidays:"]


if __name__ == "__main__":
    # Getting user input and dealing with errors caused by invalid output
    try:
        dr = get_user_input()
    except:
        print("\nInvalid Input. Exiting Program.")
```

```
        exit()

    # Computing the total number of days between start and end date
    print("\ntotal days from start date to end date:",
          compute_total_days(dr.start, dr.end))

    # Computing the total additional days from leap years
    print("\ntotal additional days from leap years:",
          compute_leap_years(dr.start, dr.end))

    # Computing the total number of weekend days
    print("\ntotal weekends:",
          compute_weekends(dr.start, dr.end))

    # Computing the total number of weekdays
    print("\ntotal days without weekends:",
          compute_weekdays(dr.start, dr.end))

    # Extra line in format
    print()

    # Computing the total number of holidays
    for holiday, count in compute_holidays(dr.start, dr.end).items():
        print(holiday, count)

    # Computing the total number of working days
    print("\ntotal working days:",
          compute_workdays(dr.start, dr.end))
```