Overview of the Software

- General Functionality: The software converts a given Gregorian date (year, month, day) into its corresponding Solar Hijri (Jalali) date.
- Problem Addressed: People who use the Solar Hijri calendar, particularly in regions like Iran and Afghanistan, often need to convert dates between the Gregorian and Solar Hijri calendars. This tool helps by providing an accurate conversion.

2. Analysis of the Software

- Information about the Problem/Objective:
- The Gregorian and Solar Hijri calendars differ in year lengths, leap years, and month lengths, making conversion complex.
- The goal is to convert Gregorian dates to Solar Hijri dates accurately by taking these differences into account.
- Requirements for the Software:
- The software should accurately convert any Gregorian date from 622 AD onward.
- It should handle leap years for both the Gregorian and Solar Hijri calendars.
- Expected Outcome: Given a Gregorian date as input, the software outputs the equivalent Solar Hijri date.

3. Algorithm Used in the Software

- Conversion Logic:
- Calculate the total number of days since a fixed start date in the Gregorian calendar (typically March 21, 622 AD).
- Use this day count to derive the corresponding Solar Hijri year, month, and day by dividing it into Solar Hijri calendar cycles.
- Pseudocode:

```
# Define arrays for the number of days in each month for both calendars
gregorian_days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
jalali_days_in_month = [31, 31, 31, 31, 31, 31, 30, 30, 30, 30, 30, 29]

# Function to check if a Gregorian year is a leap year
def is_gregorian_leap(year):
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)

# Main conversion function
def gregorian_to_jalali(g_year, g_month, g_day):
    # Step 1: Calculate total days since start date (March 21, 622 AD)
    g_day_count = calculate_days_since_start(g_year, g_month, g_day)

    # Step 2: Apply offset and calculate Solar Hijri date
```

```
    j_year = base_year + derived_value  # Base year and calculation logic
    j_month = calculate_month_from_days(day_count)
    j_day = calculate_day_in_month(remaining_days)

    return (j_year, j_month, j_day)
```

- Explanation of Approach:
- The algorithm first calculates the total days from a base Gregorian date (often the start of the Hijri calendar).
- It then applies adjustments for leap years and calendar offsets.
- Finally, it translates the day count into a Solar Hijri year, month, and day.

- Top-Down Approach:
- The solution starts by calculating total days (high-level step) and then progressively refines this calculation into year, month, and day (detailed steps).

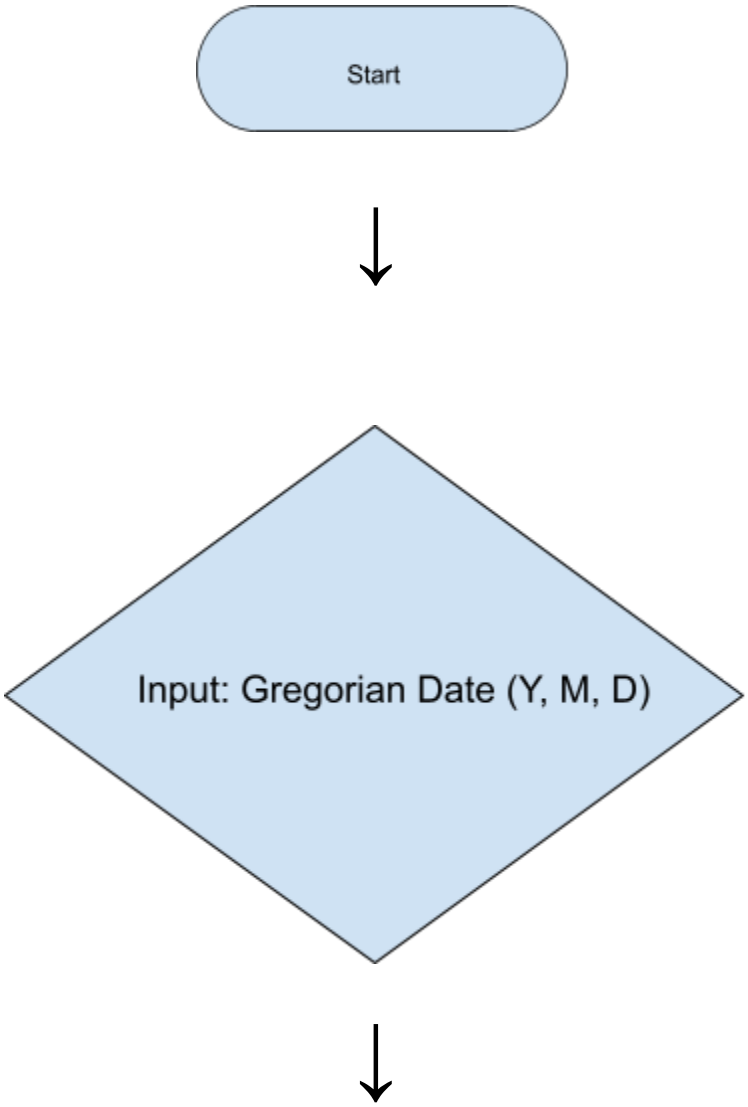
4. Implementation Details

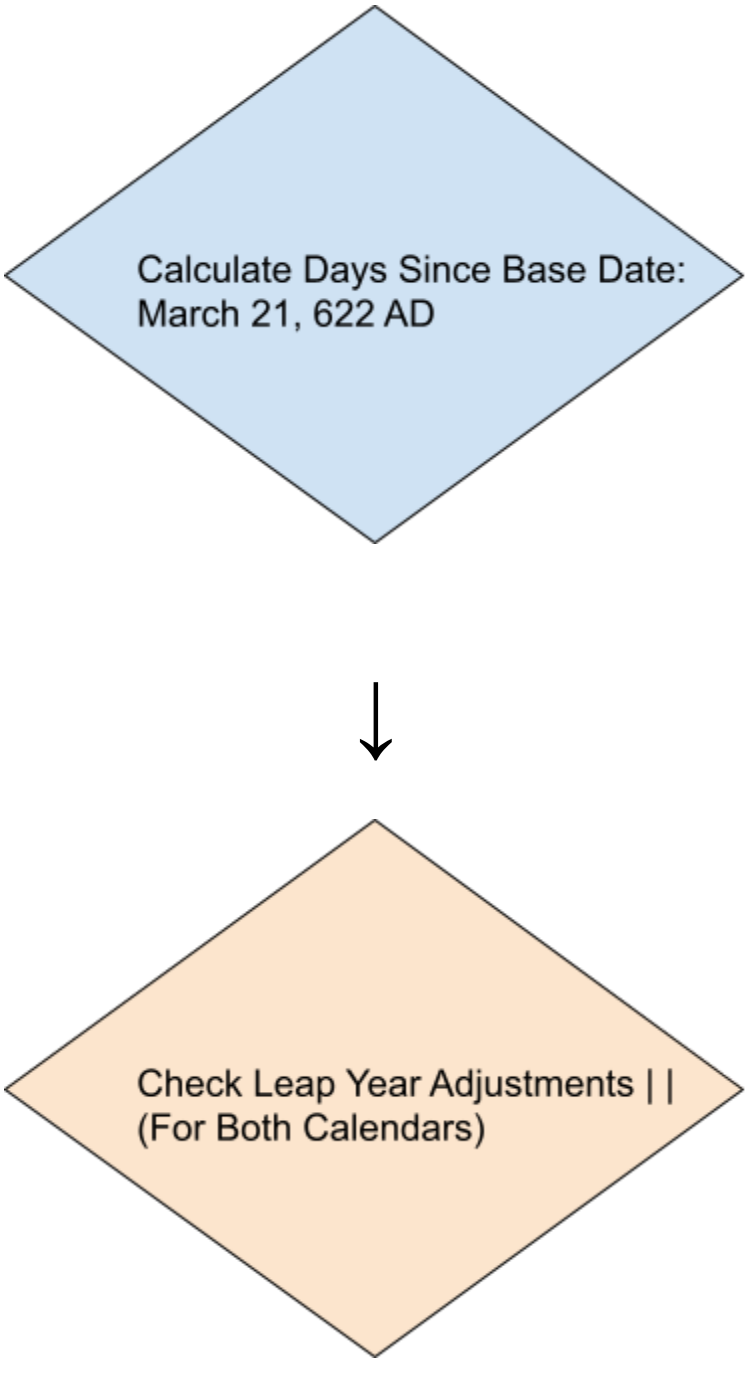
- Data Structures:
- Arrays for storing month lengths.
- Functions to check leap years.
- Programming Constructs:
- Conditionals (for leap year checks), loops (for calculating day counts), and basic arithmetic.
- Potential Challenges:
- Handling leap years correctly in both calendars.
- Ensuring data accuracy across large date ranges.

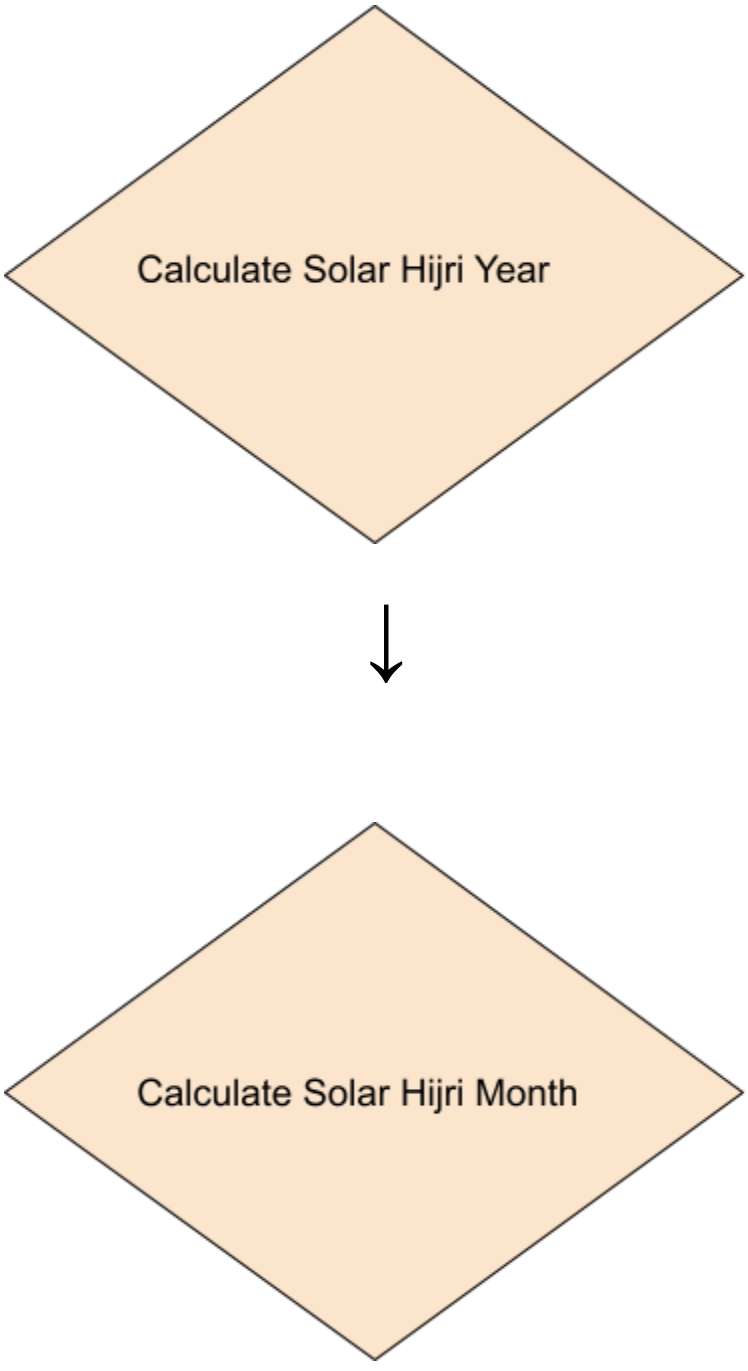
5. Example Output

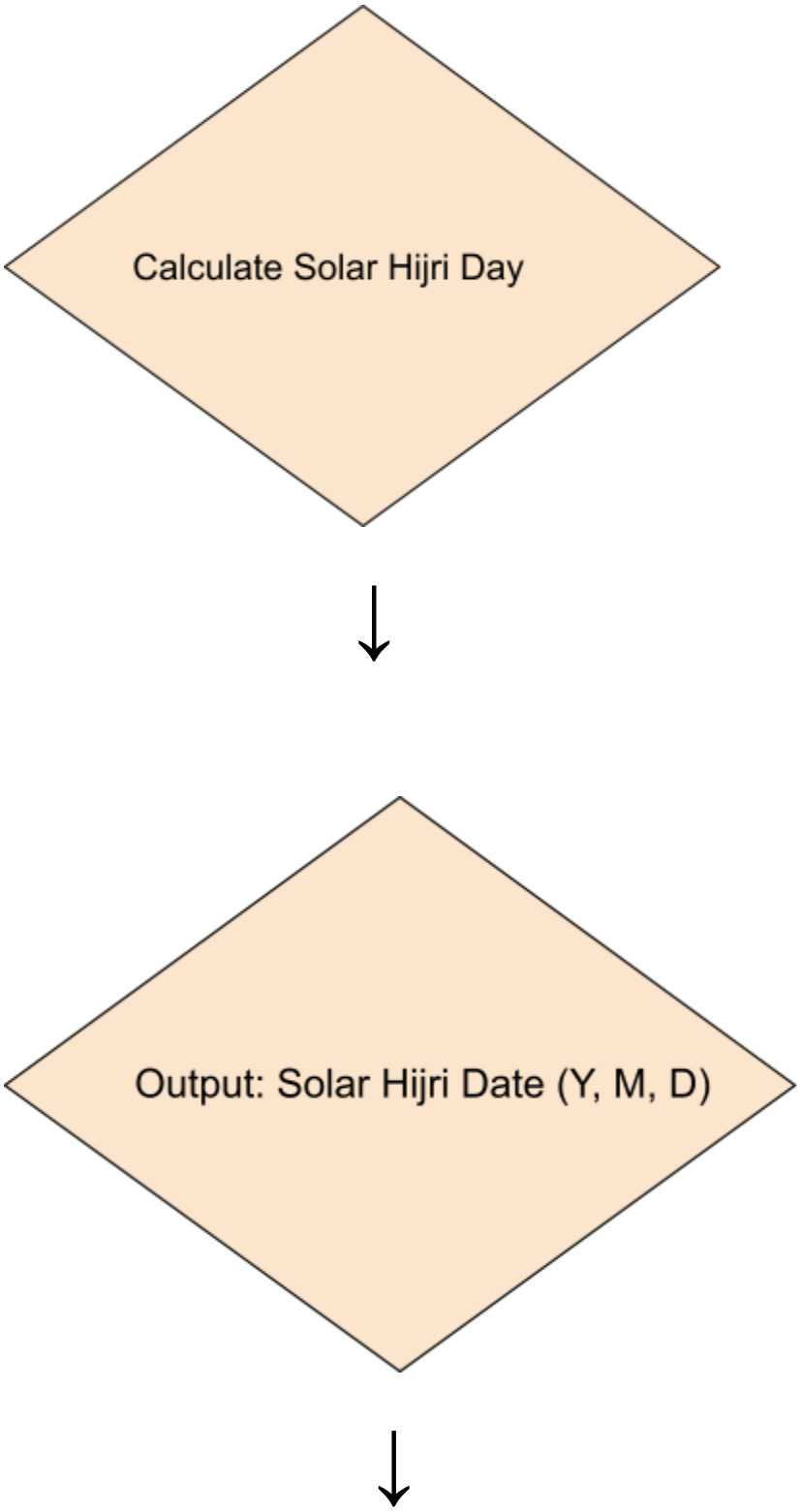
- Input: 2024-11-10
- Output: Solar Hijri Date: 1403-08-19

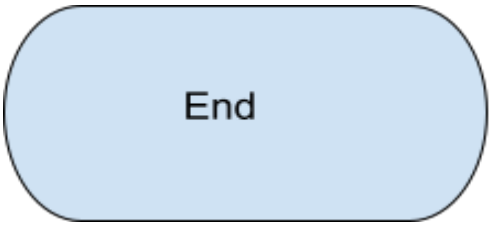
Flow chart:



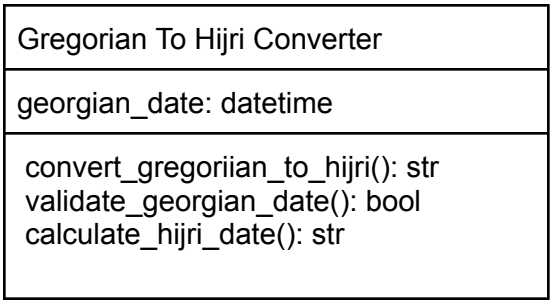

Calculate Days Since Base Date:
March 21, 622 AD
Check Leap Year Adjustments | |
(For Both Calendars)

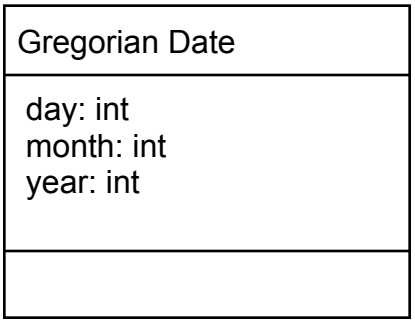

Calculate Solar Hijri Year
Calculate Solar Hijri Month

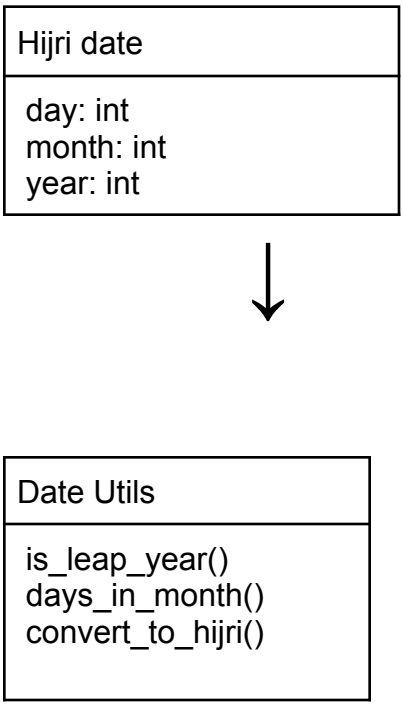

Calculate Solar Hijri Day
Output: Solar Hijri Date (Y, M, D)


End

UML Diagram:


Gregorian To Hijri Converter
georgian_date: datetime
convert_gregoriian_to_hijri(): str
validate_georgian_date(): bool
calculate_hijri_date(): str


Gregorian Date
day: int
month: int
year: int


Hijri date
day: int
month: int
year: int
Date Utils
is_leap_year()
days_in_month()
convert_to_hijri()