1185. Day of the Week

Easy Math

Problem Description

This problem requires determining the day of the week for a specific date provided by the user. The input is given as three separate integers that denote the day (`day`), the month (`month`), and the year (`year`). The expected output is a string that represents the name of the corresponding day of the week, which can be any of these: "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", or "Saturday".

Understanding how to map a specific date to a day of the week involves knowledge of calendars and possibly some algorithmic computation. Modern programming languages include date and time libraries that can make this task straightforward, but the intellectual challenge lies in calculating it manually, using a known algorithm like Zeller's Congruence.

Intuition

The intuition behind the solution is based on the understanding that dates follow a regular pattern which repeats every week. However, manual calculation of the day for a given date from scratch is complex and involves taking into account leap years, the varying number of days in each month, and adjustments for the Gregorian calendar corrections.

One direct approach is to leverage the built-in capabilities of the `datetime` library in Python that abstracts these complexities and provides a simple method to get the day of the week. The `datetime.date(year, month, day).strftime('%A')` function constructs a date object with the given year, month, and day, and then formats it as a string representing the full name of the day of the week (`'%A'`).

Additionally, the reference provided suggests using the Zeller formula as an alternative approach, which is a mathematical algorithm that can output the day of the week for any given date in the Gregorian calendar. This approach is an example of how such a problem could be solved before widespread use of computers and can be fun to implement and understand.

However, in a practical application where the algorithm's performance and simplicity are more critical, it is advisable to use built-in language features like `datetime` for reliability and efficiency, unless the problem specifically restricts their use.

Solution Approach

In the provided Python solution, the `datetime` library is utilized, which is a built-in Python module designed to work with date and time. This library takes care of the complex computations internally and provides a straightforward interface for getting the day of the week for a given date.

Implementation Steps:

1. The `datetime.date(year, month, day)` function constructs a date object from the provided day, month, and year.
2. The `.strftime('%A')` method is called on the date object to format it into a human-readable string that represents the full name of the day of the week. The `'%A'` format code tells `strftime` to extract the full weekday name.

Since the reference solution approach also hints at Zeller's formula, let's explore that method briefly:

Zeller's Congruence is an algorithm devised by Christian Zeller to calculate the day of the week for any Julian or Gregorian calendar date. It involves a formularization that takes into account the day, month, year, and some constants representing the days ("Saturday" = 0, "Sunday" = 1, "Monday" = 2, etc.).

The formula looks like this:

```
h = (q + ((13*(m + 1)) / 5) + K + (K / 4) + (J / 4) - 2 * J) % 7

Where:

h = day of the week (0 = Saturday, 1 = Sunday, 2 = Monday, ...)
q = day of the month
m = month (3 = March, 4 = April, 5 = May, ..., 14 = February)
K = year of the century (year % 100)
J = zero-based century (year / 100)
```

It's important to note that the original month numberings must be adjusted such that March is 3 and February is 14, which means January and February are considered as the 13th and 14th months of the previous year.

While Zeller's Congruence is an elegant solution, the Python `datetime` method provides a cleaner and less error-prone implementation for everyday programming needs.

It's evident that the `datetime` method is more straightforward and does not explicitly involve understanding the internal calculations of day-to-week mappings. This simplicity is why the Python `datetime` solution is often preferred for practical applications.

Example Walkthrough

Let's walk through an example to illustrate the solution approach using Python's `datetime` module. Consider that we want to find out the weekday for the date February 15, 2023. Here's how the provided solution approach would be implemented step by step:

Step 1: Construct the Date Object

We start by creating a date object in Python for the given date — in this case, February 15, 2023. We use the `datetime.date` function, passing in the year `2023`, the month `2`, and the day `15`.

```
import datetime
date_object = datetime.date(2023, 2, 15)
```

The `date_object` now holds this specific date and allows us to perform operations on it, such as formatting to day names.

Step 2: Format and Get the Weekday

Now that we have a date object, we want to find out what day of the week this date falls on. We use the `.strftime('%A')` method on `date_object` to format it as a string that represents the full name of the day of the week.

```
weekday_name = date_object.strftime('%A')
print(weekday_name)  # Output should be "Wednesday"
```

The output of printing `weekday_name` should be `"Wednesday"`, since February 15, 2023, indeed fell on a Wednesday.

This example demonstrates the simplicity of using Python's `datetime` library to determine the day of the week for a given date with minimal code. It abstracts all the complex calculations and adjustments that might otherwise be necessary, such as handling leap years, the irregular number of days in months, and century leap adjustments.

In contrast to manual methods like Zeller's Congruence, which can be a more mathematical and intriguing exercise to implement, using Python's built-in `datetime` features provides a faster and more readable way to achieve the same result in everyday practical programming scenarios.

Solution Implementation

Python

```
import datetime  # Ensure datetime module is imported

class Solution:
    def day_of_the_week(self, day: int, month: int, year: int) -> str:
        # This method takes a day, month, and year, and returns the name of the day of the week.

        # Create a date object with the provided year, month, and day.
        date_object = datetime.date(year, month, day)

        # Return the full weekday name (e.g., 'Monday', 'Tuesday', etc.) as a string.
        return date_object.strftime('%A')
```

Java

```
import java.util.Calendar;

public class Solution {

    // Array containing the days of the week for easy reference.
    private static final String[] DAYS_OF_WEEK = {
        "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"
    };

    /**
     * Returns the day of the week for a particular date.
     *
     * @param day   The day of the month (1-based).
     * @param month The month of the year (1-based, January is 1).
     * @param year  The year (e.g., 2021).
     * @return The name of the day of the week.
     */
    public static String dayOfTheWeek(int day, int month, int year) {
        // Obtain an instance of Calendar.
        Calendar calendar = Calendar.getInstance();

        // Set the calendar to the specified date.
        // The month is zero-based in Calendar, hence the need to subtract 1.
        calendar.set(year, month - 1, day);

        // Get the day of the week from the Calendar (1 = Sunday, 7 = Saturday)
        // and adjust by subtracting 1 to align with the index of DAYS_OF_WEEK array.
        int dayOfWeekIndex = calendar.get(Calendar.DAY_OF_WEEK) - 1;

        // Return the string representation of the day of the week.
        return DAYS_OF_WEEK[dayOfWeekIndex];
    }
}
```

C++

```
#include <vector>
#include <string>

class Solution {
public:
    // Function to find the day of the week for a given date
    string dayOfTheWeek(int day, int month, int year) {
        // Zeller's congruence algorithm adjustments for January and February
        if (month < 3) {
            month += 12;
            year -= 1;
        }

        // Extracting the century and year of the century
        int century = year / 100;
        year %= 100;

        // Zeller's congruence formula to calculate the day of the week
        int weekDay = (century / 4 - 2 * century + year + year / 4 + 13 * (month + 1) / 5 + day - 1) % 7;

        // Array of week days starting with Sunday
        vector<string> weekDays = {
            "Sunday", "Monday", "Tuesday", "Wednesday",
            "Thursday", "Friday", "Saturday"
        };

        // Adjusting for negative values and returning the result
        return weekDays[(weekDay + 7) % 7];
    }
};
```

TypeScript

```
// Define an array of week days starting with Sunday
const weekDays: string[] = [
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday"
];

// Function to find the day of the week for a given date
function dayOfTheWeek(day: number, month: number, year: number): string {

    // Zeller's congruence algorithm adjustments for January and February
    if (month < 3) {
        month += 12;
        year -= 1;
    }

    // Extracting the century and year of the century
    const century: number = Math.floor(year / 100);
    year = year % 100;

    // Zeller's congruence formula to calculate the day of the week
    let weekDay: number = (century / 4 - 2 * century + year + Math.floor(year / 4) + 13 * (month + 1) / 5 + day - 1) % 7;

    // Adjusting for negative values and returning the result
    weekDay = (weekDay + 7) % 7;
    return weekDays[weekDay];
}

// Usage example:
// const day: string = dayOfTheWeek(15, 8, 2021);
// console.log(day); // Outputs the day of the week for August 15, 2021
```

```
import datetime  # Ensure datetime module is imported

class Solution:
    def day_of_the_week(self, day: int, month: int, year: int) -> str:
        # This method takes a day, month, and year, and returns the name of the day of the week.

        # Create a date object with the provided year, month, and day.
        date_object = datetime.date(year, month, day)

        # Return the full weekday name (e.g., 'Monday', 'Tuesday', etc.) as a string.
        return date_object.strftime('%A')
```

Time and Space Complexity

The time complexity of the provided code snippet is `O(1)`. This is because it utilizes Python's built-in `datetime` module to find the day of the week for a given date. This operation does not depend on the size of the input and takes constant time.

The space complexity is also `O(1)`. The function only creates a single `date` object and formats it, not using any additional space that scales with the input size.