# Exercise: Calendar trick

## The trick

If someone tells you a date, it is possible (but fiddly) to tell them the day of the week for that date.

Here we follow a scheme developed by Lewis Carroll (author of Alice's Adventures in Wonderland), published in *Nature* on 31 March 1887 (a Thursday).

We must calculate several parts

- $C$: a century number.
    - For dates after 2 September 1752, we use take the remainder after dividing by 4 from 3 and double it. For example, for years starting '20', we have `20 % 4 = 0` so do $(3-0)\times 2 = 6$. For years starting '19', `19 % 4 = 3` so do $(3-3)\times 2 = 0$.
    - For dates until 2 September 1752 (do you know why these are different?), we simply subtract the number from 18. For example, for 1676 we take $18-16=2$.
- $Y$: a year number. We divide the two-digit year by 12, and add the number of 12s to the remainder. Then we divide the remainder by 4 and add the number of 4s. So for example for 2030, the year is '30'. We know $30 = 2\times 12 + 6$ and $6 = 1\times 4 + 2$, so we add $2+6+1=9$.
- $M$: a month number. January is zero. To calculate later months, add the number of days in each module (mod 7), so for example the number for February is $0+31 = 3 \pmod 7$. Then March is $3+28 = 3 \pmod 7$, April is $3+31 = 6 \pmod 7$, and so on. (We'll deal with leap years separately.)
- $D$: simply the day number in this month.

The answer is $C+Y+M+D \pmod 7$, except you subtract $1 \pmod 7$ if the date is in January or February of a leap year. The number you get is the day of the week, with 0=Sunday, 1=Monday, and so on.

## Writing a program

If you do the program in your head as a party trick, there are some shortcuts that help you calculate the items more quickly. Computer programs work quickly anyway, so we don't need these.

### Enter a date

First, we want the user to enter a date using `input()`.

```
print("Please think of a date")
day = int(input("Please enter the day of the month (1, 2, 3, etc.): "))
```

```
month = int(input("Please enter the month (1=Jan, 2=Feb, etc.): "))
fullyear = input("Please enter the year (2025, etc.): ")
```

Now we must break the `fullyear` into two parts.

```
century = int(fullyear[0:2])
year = int(fullyear[2:])
```

We can check this by printing it all out.

```
print(f"day: {day}, month: {month}, century: {century}, year: {year}")
```

## Day number

Let's start with the day number, because it's easy. Our day number `D` is the same as the number the user entered.

```
D = day
```

## Century number

First we need to know whether the current date is under the old or new system. The number the user entered is stored as `century`, so we can test using this.

This is slightly fiddly, because if we're in September 1752 we want to know if it is before or after the 2nd, otherwise if we're in 1752 we want to know if we're before or after September, or if we're in the 1700s are we before or after the 52nd year. We do this with a combination of `and` and `or`, with brackets to make the order operate properly.

```
if (century == 17 and year == 52 and month == 9 and day <= 2) or (century == 1
    print("old system")
else:
    print("new system")
```

Now, where we printed "old system", we need some code to calculate the century number. This is simply the variable `century` subtracted from 18.

```
C = 18 - century
```

Where we printed "new system" is more complicated.

First we find the remainder from dividing by 4.

```
rem = century % 4
```

Now we take this number from 3 and double it.

```
C = (3 - rem) * 2
```

This is `C`, the century number, which we will take forward to our later calculations.

## Year number

First we find the remainder when dividing the user's number `year` by 12.

```
rem = year % 12
```

Now we want to find how many times 12 divides into `year`. We use `int` to truncate this

```
twelves = int(year/12)
```

Finally, we need the number of times `rem` divides by 4.

```
fours = int(rem/4)
```

Now we add these numbers together to obtain the year number, `Y`.

```
Y = twelves + rem + fours
```

## Month number

We can work out all the month numbers and store them in a list. First we define an empty list, `month_numbers`.

```
month_numbers = []
```

For January, the month number is zero. We need a loop that stores this then works out the month number for the following month by repeatedly adding the number of days in each month and store this number in our list `month_numbers`.

```
thismonth = 0
for n in range(1,13):
    month_numbers.append(thismonth)

    if n in (1,3,5,7,8,10,12): # Jan, Mar, May, Jul, Aug, Oct, Dec
        thismonth += 31
    elif n in (4,6,9,11): # Apr, Jun, Sep, Nov
        thismonth += 30
    else: # Feb
        thismonth += 28

    thismonth = thismonth % 7
```

If you `print(month_numbers)` you can check whether this has worked correctly.

Now for the date the user entered, we can simply read off the month number.

Warning: Python numbers items in a list starting with 0 for January, whereas your user will have entered 1 for January. The number the user entered is `month`, so we want to find the `month-1` item in our list.

```
M = month_numbers[month-1]
```

## Calculating the day of the week

Now we have calculated `C`, `Y`, `M` and `D`. We can print these out to check their value.

```
print(C,Y,M,D)
```

The day of the week is given by the sum of these digits modulo 7.

```
w = (C + Y + M + D) % 7
```

Remember Lewis Carroll told us that 0 is Sunday, 1 is Monday and so on. This lends itself very well to using a list of day names.

```
weekdays = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
print(weekdays[w])
```

There's one last catch to deal with - if the date is in January or February of a leap year, we subtract 1

```
w = (C + Y + M + D - 1) % 7
```

Can you use your code to work out whether this is a leap year to sort this last step? Remember that the year is stored as `fullyear` and you will need to turn it into an integer.

Now you have a piece of code that you can use to find the week day of any date. Use this to check that Lewis Carroll's 31 March 1887 article was published on a Thursday. Carroll gave two examples in his article. Use your program to check today's date and these two:

- 18 September 1783, Thursday
- 23 February 1676, Wednesday

Now you know your code is working, it's traditional at this point to use the code to find the weekday on which you were born and other important dates in your life and in history.