New Year's Day on Saturday

The USSR Olympiad Problem Book, by D. O. Shklarski, N . N. Chentzov, and I. M. Yaglom.

#14.* (a) On which of the two days of the week, Saturday or Sunday, does New Year's Day fall more often?

Proof. The temptation is to say that New Year's day is equally likely to fall on a Saturday or Sunday, but that's not so. The reason has to do with the vagaries of the Gregorian calendar, first established in 1582 to reconcile the earth's rotation on its axis with its rotation around the sun. Julius Caesar established the Julian calendar in 45 BC, according to which leap years occurred every four years without fail, but that makes the year just a little too long in terms of days (1 year = $365^1/_4$ days). The closer value, which changes slightly over time, is 365.24219 days, the length of a mean tropical year. Pope Gregory's mathematicians brilliantly approximated the mean tropical year by removing three leap years every 400 years, making one year equal $365^{97}/_{400} = 365.2425$ days, closer to the real value by a factor of 30. The leap years removed are the century years not a multiple of 100, so 1700, 1800, and 1900 were *not* leap years but 2000 still was.

The Gregorian calendar cycles perfectly every 400 years. To see this, note that 400 years = 400 * 365 + 97 = 146,097 days, the 97 because of the extra day in every leap year. But 146,097 = 20,871, so exactly 20,871 weeks have transpired over those 400 years, and the next 400-year cycle starts on the same day of the week as the last one. The century years are distributed over the new cycle in the same pattern as they were over the previous cycle, so the days of the week proceed in the new cycle in exactly the same pattern they did in the last one. This all implies that the analysis for a problem like the one at hand can be restricted to one 400-year cycle and it doesn't matter where the cycle starts. Right away this implies that the days of the week Jan 1 falls on are not equally distributed because there are $12 \cdot 400 = 4800$ months in a cycle, which is not a multiple of 7.

So start the cycle on New Year's Day the year this was written, namely Jan 1, 2024, which was a Monday. Let Sunday = 0, Monday = 1, Tuesday = 2, ... Saturday = 6, and chart the day of the week January 1 falls on for years following 2024 (the 1 under 2024 indicates that Jan 1, 2024 was a Monday, for example):

2024	2025	2026	2027	2028	2029	2030	2031	2032	2033
1	3	4	5	6	1	2	3	4	6

Generally the day for Jan 1 advances by 1 mod 7, except right after leap years, when it advances by 2 mod 7 (the days in red). Consider the 28 year mini-cycle 2024 – 2051, which starts as shown in the chart. The missing days are 2 between 2024 and 2025, 0 between 2028 and 2029, 5 between 2032 and 2033, and so on for those not shown. Write out the sequence of values in the second row, but inserting the skipped values (the inserted values are bolded):

$$1 \quad \mathbf{2} \quad 3 \quad 4 \quad 5 \quad 6 \quad \mathbf{0} \quad 1 \quad 2 \quad 3 \quad 4 \quad \mathbf{5} \quad 6 \quad \ldots$$

This sequence is a string of 35 consecutive integers mod 7, so it takes every value mod 7 exactly 5 times. But the bolded numbers are all different, namely 0, 1, 2, 3, 4, 5, 6 in some order. To see this, note that the bolded numbers advance by 5 mod 7 and a sequence of numbers advancing by 5 mod 7 one after the other represent different equivalence classes mod 7 because 5 and 7 are relatively prime. It follows that the bolded numbers are exactly 0 through 6 in some order. Deleting them from the string of 35 consecutive integers results in 28 integers evenly distributed among the equivalence classes mod 7. These are exactly the days that Jan 1 falls on in the 28 year mini-cycle, implying that Jan 1 falls on each day of the week exactly 4 times.

The analysis above applies to *every* 28 year mini-cycle that does not contain an exceptional century year, one of 2100, 2200, or 2300 in the 400-year cycle $2024 - 2423$ being discussed. Therefore another mini-cycle for $2052 - 2079$ can be brought in to show that Jan 1 falls on each day of the week exactly 8 times between 2024 and 2079. The next 28-year cycle $2080 - 2107$ has to be handled separately because it contains the exceptional year 2100, and in fact the days of the week Jan 1 falls on are *not* equally distributed over this mini cycle, which is generally the case for 28 year mini-cycles including one of the exceptional century years. Normality returns over the period $2108 - 2191$, which contains each day of the week 12 times, followed by the abnormal 28-year mini-cycle containing 2200.

This can go on until the final solution is found, which is essentially how the solution in the book proceeds (for two pages!) and was all that could be done in 1935. In 2024 though, we've got Python with its rich date-handling facilities, and this does the trick:

```
import datetime

# Init array with 0s.
newYearsDays = [0]*7

for year in range(2024, 2424):
  # Get datetime object for Jan 1 of the year.
  d = datetime.date(year, 1, 1)
  # Get its day of the week -- adjust from Python's
  #  0 for Monday to our 0 for Sunday.
  dayOfWeek = (d.weekday() + 1) % 7
  # Load array with each slot representing the number
  #  of times that day of the week is a Jan 1.
  newYearsDays[dayOfWeek] += 1

# [58, 56, 58, 57, 57, 58, 56]
print(newYearsDays)
```

The 0th element of the array establishes that Sunday occurs as New Year's Day 58 times in a 400-year cycle, the last element that Saturday occurs 56 times, so Sunday occurs more often than Saturday. **QED.**

(b) On which day of the week does the thirtieth of the month most often fall?

Proof. The solution in the book gives the answer succincly with the prologue "Employing methods analogous to those used in problem (a), we can show that ...". The development would be considerably more intricate though, the analysis having to track which day of the week Jan 1 falls on and then the pattern for the months of that year (and *that* will vary depending on whether it's a leap year or not). Python to the rescue:

```
import datetime

# Init array with 0s.
thirtiethDays = [0]*7

for year in range(2024, 2424):
  for month in range(1, 13):
    # No 30th in February!
    if (month != 2):
      # Get datetime object for the 30th of the month.
      d = datetime.date(year, month, 30)
      # Get its day of the week -- adjust from Python's
      #  0 for Monday to our 0 for Sunday.
      dayOfWeek = (d.weekday() + 1) % 7
      # Load array with each slot representing the number of
      #  times that day of the week is the 30th of some month.
      thirtiethDays[dayOfWeek] += 1

# [627, 631, 626, 631, 627, 629, 629]
print(thirtiethDays)
```

Element 0 of the array indicates that the thirtieth of the month occurs on Sunday 627 times, element 1 that the thirtieth is on a Monday 631 times, and so on, so Mondays and Wednesdays are tied for the most occurences of the thirtieth of the month at 631. Note the sum of the array elements = $627+631+\cdots+629 = 4400$, which equals 400 years of 11 months each (no February).

The answer in the book is incorrect:

```
[687, 685, 685, 687, 684, 688, 684]
```

These are the correct answers for the number of days of the week falling on the *thirteenth* of the month (note they add up to 4800). I'm guessing a mistranslation of thirteenth to thirtieth, close in Russian as they are in English. **QED.**

– Mike Bertrand
  May 6, 2024