Perl script reveals math trick

MIND GAMES

Aleksandr Popov, Fotolia

A trick that anybody can learn lets you determine the day of the week from the date. We'll apply some Perl technology to discover whether the method is reliable.

BY MICHAEL SCHILLI

Recently, I was reading *Mind Performance Hacks* [1] and stumbled across Hack Number 43, which explains how to determine the day of the week for any given date.

The method is attributed to Lewis Carroll [2], the author of *Alice's Adventures in Wonderland*. Simply calculate four values, one after the other, for the year, month, and day, and a fourth value as an adjustment factor. Then add the values, divide by 7, and the remainder amazingly gives you the day of the week as a number between zero (Sunday) and six (Saturday).

Example Day

As an example, I'll take the day I wrote this article, October 4, 2007, as a starting point. To obtain the year value, do the following:

```
(YY + (YY div 4)) mod 7
```

Table 1: Month Values

Value	Month
0	January
3	February
3	March
6	April
1	May
4	June
6	July
2	August
5	September
0	October
3	November
5	December

where YY is the two-digit year (07 for 2007). The *div* operator divides without a remainder; 7 div 4 is thus 1 because 7 divided by 4 gives you a result of 1, and the remainder of 3 is ignored. A modulo 7 operation is performed on the resulting sum, and 8 modulo 7 is 1. The value for the year is thus 1.

Table 2: Adjustment Values for Years

Year	Value
1700	4
1800	2
1900	0
2000	6
2100	4
2200	2
2300	0

Table 3: Day Values

Value	Day of the Week
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

Table 4: Examples

Date	Year value	Month value	Day value	Adjustment	Day of the Week
01.01.1970	3	0	1	0	4 Thursday
14.07.1995	6	6	14	0	5 Friday
11.09.2001	1	5	11	6	2 Tuesday
01.02.2004	5	3	1	5	0 Sunday
01.03.2004	5	3	1	6	1 Monday
04.10.2007	1	0	4	6	4 Thursday

The month value is taken from Table 1, which you need to memorize using a mnemonic device (described later in the "Mnemonic Tricks" section). The table gives a month value of 0 for October. The value for the day is simply its number, so October 4 has a day value of 4.

The fourth value to work out in your head is taken from Table 2, which gives an adjustment value of 6 for years between 2000 and 2099.

Memorizing this table is not necessary; just remember the values of 2000 (6) and 1900 (0) because most people will ask for days of recent events or birth dates. One caveat is that you need to subtract 1 for leap years if the date you are looking for is in January or February, but 2007 isn't a leap year so you can ignore this for now.

The four values are 1 (year), 0 (month), 4 (day), and 6 (adjustment factor). If you add these values (=11) and perform a modulo 7 operation on the result, you get a value of 4.

A quick glance at Table 3 reveals that – roll of drums, please! – October 4, 2007, was a Thursday.

Testing, One, Two!

The question is whether this trick really does give you the day of the week for any date. The script in Listing 1 [3] iterates through every single day from 1.1.1700, (day.month.year) to the present day – through a period of history that saw the California Gold Rush and two world wars – and into the future world of "Star Trek: The Next Generation" and the 24th century.

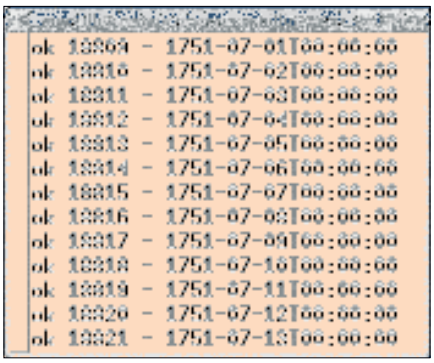

Figure 1: Test::More outputs the results of individual tests.

Line 34 of the script defines the function *wday_mindcal()*, which expects the four-digit year, month, and day for a date. Then the script applies the rules explained above to calculate the values for the year/month/day and an adjustment factor, and performs the calculations required to determine the number of the day of the week.

The month values are stored in the *@MONTH* array, from January to December. Because arrays start on index 0 rather than 1, you must subtract 1 from the month you are looking for to access the array with the correct index.

The adjustment factors for the various centuries are stored in the *%ADJ* hash. Lines 10 through 12 populate the hash by assigning a list that alternately has the century numbers and the matching adjustment values.

The reference values are calculated by the CPAN *DateTime* module, which counts upward from 1.1.1700 through 31.12.2399 and provides the day, month, and year for each iteration.

The infinite loop starting in line 20 isn't really infinite because line 30 exits if the year for the current date is higher than 2399. Inside the loop, the *DateTime* object's *add()* method adds a day to the current date and sends the program into the next iteration.

CPAN for Comparison

DateTime has a day of the week function, *wday()*, which numbers the days from 1 (Monday) through 7 (Sunday). Mindcal does a modulo 7 to change the 7 for Sunday into a zero and compares the results with the values returned by *wday_mindcal()*, which will be between 0 (Sunday) and 6 (Saturday).

The script implements a remainderless division function, *div*, by enabling a pragma in line 39, *use integer*. Once you set this mode, Perl will ignore floating points and work with integers, meaning that 17/4 gives you a straight 4.

The *leap_year()* function in line 64 determines whether the year is a leap year. If the year is divisible by 4, it is a leap year, unless it is divisible by 100. On the other hand, if it is divisible by 400, it is a leap year. Of course, *DateTime* has the same logic, but you want the script to emulate somebody working this out mentally.

The comparisons are performed by the *is()* function exported by the CPAN *Test::More* module. The function expects three parameters: the actual value provided by the math genius in the mindcal script, the reference value from *DateTime*, and a comment comprising a stringified – and thus readable – *DateTime* object.

The script passes *no_plan* in to the *Test::More use* instruction to specify that the number of tests to be performed is undefined.

Figure 1 shows the output from mindcal. Working on the assumption that nobody is interested in watching 250,000 lines of output scroll down the screen, the *Test::Harness* module neatly collates the results provided by the test suite. The command line

```
perl -MTest::Harness ➥
-e'runtests("mindcal")'
```

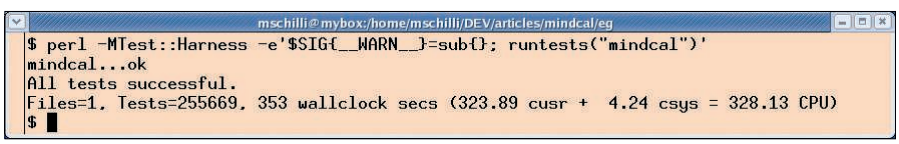

Figure 2: Test::Harness collates the results.

runs the mindcal script and monitors which tests return values of *ok* and which return *not ok*.

Test Results

At the end of the test, the user is given a neat summary, as shown in Figure 2. The command line in Figure 2 is slightly different from that above. *Test::Harness* has the unpleasant habit of complaining on STDERR if you have more than 100,000 test cases in a suite.

The command line shown in Figure 2 drops these messages into a black hole via the *__WARN__* signal handler. Lo and behold, Lewis Carroll was right – all 255,669 tests run without error.

Mnemonic Tricks

A memory aid can help you remember the month values. The numbers from January through December are 033614625035 if you lump them all together. Preferably, I split this monster number up into groups of three, 033-614-625-035, and then use the mnemonic: "Start with 033 and go to 614 like 3.14 – easy as pi, but with a 6 at the start, because we just had two 3s. Follow with 625; it starts with a 6 and is 25^2, the area of a square – a nice change from the circle (pi) we just had. Finish with 035, two more than the first group of 033."

I visualize the groups of three, which makes it easy to jump to the start of the second half of the year group, with the number 6 is assigned to July.

Practice Makes Perfect

At first, your calculations might go quite slowly, especially if you are trying to work out the days of the week in the latter part of last century.

Think about 1995, for example: 95 divided by 4 equals 23, ignoring fractions; 95 plus 23 equals 118, and 118 modulo 7 is 6. If you can't do this quickly in your head, you should stick to accepting dates from a specific year or from the current year. In advance, you can calculate the year value and just remember it, for example, 1 for 2007.

More Examples

Table 4 gives you a couple of practice examples. Remember that 2004 is a leap year, so you need to deduct a point for dates in January/February, but not for the other months.

On with the Show

After you have mastered the trick, you can astound your audience with your mental prowess. Start with a small group of friends before you move on to parties and sold-out theaters! ■

Listing 1: mindcal

```
01 #!/usr/bin/perl
02 use strict;
03 use warnings;
04 use Test::More qw(no_plan);
05 use DateTime;
06
07 my @MONTH =
08  qw(0 3 3 6 1 4 6 2 5 0 3 5);
09 my %ADJ =
10   qw(1700 4 1800 2 1900 0
11      2000 6 2100 4 2200 2
12      2300 0);
13
14 my $dt = DateTime->new(
15   year  => 1700,
16   month => 1,
17   day   => 1,
18 );
19
20 while (1) {
21   my $calc =
22     wday_mindcal( $dt->year,
23     $dt->month, $dt->day );
24
25   is( $calc, $dt->wday() % 7,
26     "$dt" );
27
28   $dt->add( days => 1 );
29
30   last if $dt->year() > 2399;
31 }
32
33 ##################################
34 sub wday_mindcal {
35 ##################################
36   my ( $year, $month, $day )
37     = @_;
38
39   use integer;
40
41   my $year2 = $year % 100;
42   my $cent  = $year / 100;
43   my $y     = (
44     $year2 + ( $year2 / 4 ) )
45     % 7;
46
47   my $m =
48     $MONTH[ $month - 1 ];
49   my $d = $day;
50
51   my $adj =
52     $ADJ{ $cent * 100 };
53
54   $adj--
55     if leap_year($year)
56     and $month <= 2;
57
58   return (
59     ( $y + $m + $d + $adj ) %
60       7 );
61 }
62
63 ##################################
64 sub leap_year {
65 ##################################
66   my ($year) = @_;
67
68   return 0 if $year % 4;
69   return 1 if $year % 100;
70   return 0 if $year % 400;
71   return 1;
72 }
```

INFO

[1] Hale-Evans, Ron. *Mind Performance Hacks*. O'Reilly, 2006, *http://www.oreilly.com/catalog/mindperfhks/*

[2] Lewis Carroll's Algorithm for finding the day of the week for any given date, *http://www.cs.usyd.edu.au/~kev/pp/TUTORIALS/1b/carroll.html*

[3] Listings for this article: *http://www.linuxpromagazine.com/Magazine/Downloads/87/*

THE AUTHOR

Michael Schilli works as a Software Developer at Yahoo!, Sunnyvale, California. He wrote "Perl Power" for Addison-Wesley and can be contacted at *mschilli@perlmeister.com*. His homepage is at *http://perlmeister.com*.