2023

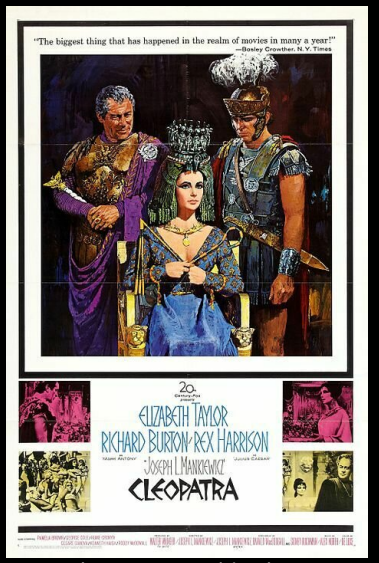
Speeding Date

Implementing Fast Calendar Algorithms

Cassio Neri

C++ now

Cleopatra (1963) by Joseph Mankiewicz



Howard Terpning, Public domain, via Wikimedia Commons.

Soundtrack: Alex North.

Julius Caesar (Rex Harrison)

Sosigenes (Hume Cronyn)

Cleopatra (Elizabeth Taylor)

SPEEDING DATE

Implementing Fast Calendar Algorithms

Cassio Neri (Independent Researcher)

Joint work with Prof. Lorenz Schneider (EMLyon, France)

Digital clocks

Digital clocks

05-10 2023

14:30 00

WED

#seconds

1683729000

since 1970 - 01 - 01 @ 00:00:00

Digital clocks

05-10 2023

14:30 00

WED

#seconds

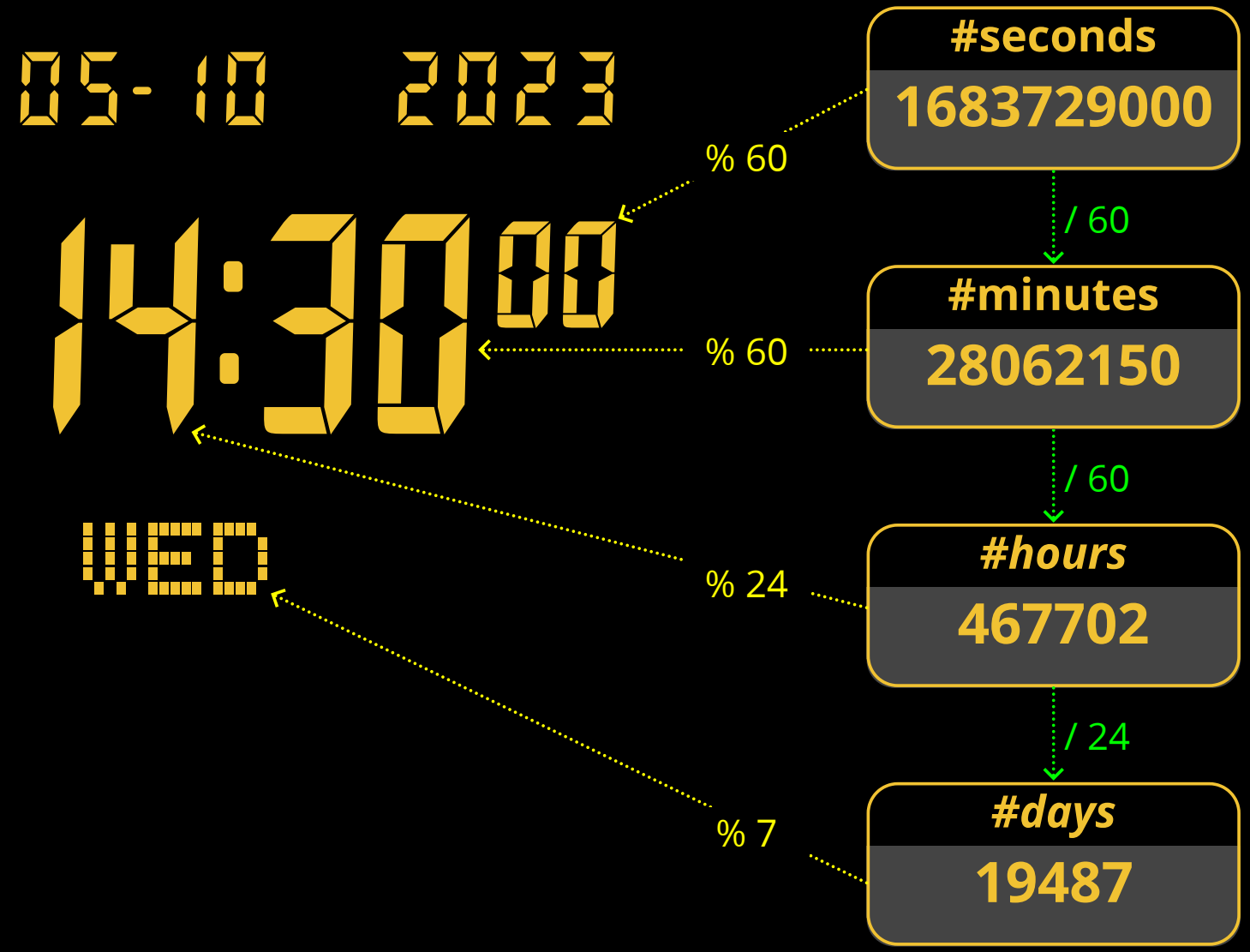
1683729000

since 1970 - 01 - 01 @ 00:00:00

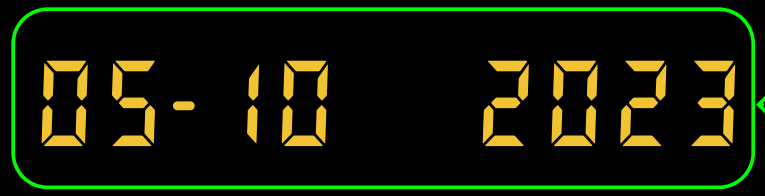
Unix epoch

Ken Thompson (sitting) and Dennis Ritchie at PDP-11.
Peter Hamer, CC BY-SA 2.0, via Wikimedia Commons.

Digital clocks



Digital clocks



?

#days
19394

since 1970 - 01 - 01 @ 00:00:00

Two solutions

.NET 6 (edited)

```
date to_date(int n) { // Epoch is 0001-Jan-01

  int s_daysToMonth365[] = {
    0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };
  int s_daysToMonth366[] = {
    0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 };

  int y400 = n / 146097;
  n -= y400 * 146097;
  int y100 = n / 36524;
  if (y100 == 4)
      y100 = 3;
  n -= y100 * 36524;
  int y4 = n / 1461;
  n -= y4 * 1461;
  int y1 = n / 365;
  if (y1 == 4)
      y1 = 3;
  int year = y400 * 400 + y100 * 100 + y4 * 4 + y1 + 1;
  n -= y1 * 365;
  bool leapYear = y1 == 3 && (y4 != 24 || y100 == 3);
  int* days = leapYear ? s_daysToMonth366 : s_daysToMonth365;
  int m = (n >> 5) + 1;
  while (n >= days[m])
      m++;
  month_t month = m;
  int day = n - days[m - 1] + 1;
  return { year, month, day };
}
```

Two solutions

.NET 6 (edited)

```
date to_date(int n) { // Epoch is 0001-Jan-01

  int s_daysToMonth365[] = {
    0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };;
  int s_daysToMonth366[] = {
    0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 };;

  int y400 = n / 146097;
  n -= y400 * 146097;
  int y100 = n / 36524;
  if (y100 == 4)
      y100 = 3;
  n -= y100 * 36524;
  int y4 = n / 1461;
  n -= y4 * 1461;
  int y1 = n / 365;
  if (y1 == 4)
      y1 = 3;
  int year = y400 * 400 + y100 * 100 + y4 * 4 + y1 + 1;
  n -= y1 * 365;
  bool leapYear = y1 == 3 && (y4 != 24 || y100 == 3);
  int* days = leapYear ? s_daysToMonth366 : s_daysToMonth365;
  int m = (n >> 5) + 1;
  while (n >= days[m])
      m++;
  month_t month = m;
  int day = n - days[m - 1] + 1;
  return { year, month, day };
}
```

Two solutions

.NET 6 (edited)

```
date to_date(int n) { // Epoch is 0001-Jan-01
```

```
  int s_daysToMonth365[] = {
    0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };
  int s_daysToMonth366[] = {
    0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 };

  int y400 = n / 146097;
  n -= y400 * 146097;
  int y100 = n / 36524;
  if (y100 == 4)
      y100 = 3;
```

```
  n -= y100 * 36524;
  int y4 = n / 1461;
  n -= y4 * 1461;
  int y1 = n / 365;
  if (y1 == 4)
      y1 = 3;
```

```
  int year = y400 * 400 + y100 * 100 + y4 * 4 + y1 + 1;
  n -= y1 * 365;
  bool leapYear = y1 == 3 && (y4 != 24 || y100 == 3);
  int* days = leapYear ? s_daysToMonth366 : s_daysToMonth365;
  int m = (n >> 5) + 1;
  while (n >= days[m])
      m++;
```

```
  month_t month = m;
  int day = n - days[m - 1] + 1;
  return { year, month, day };
}
```

Two solutions

.NET 6 (edited)

```
date to_date(int n) { // Epoch is 0001-Jan-01

  int s_daysToMonth365[] = {
    0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };
  int s_daysToMonth366[] = {
    0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 };

  int y400 = n / 146097;
  n -= y400 * 146097;
  int y100 = n / 36524;
  if (y100 == 4)
      y100 = 3;
  n -= y100 * 36524;
  int y4 = n / 1461;
  n -= y4 * 1461;
  int y1 = n / 365;
  if (y1 == 4)
      y1 = 3;
  int year = y400 * 400 + y100 * 100 + y4 * 4 + y1 + 1;
  n -= y1 * 365;
  bool leapYear = y1 == 3 && (y4 != 24 || y100 == 3);
  int* days = leapYear ? s_daysToMonth366 : s_daysToMonth365;
  int m = (n >> 5) + 1;
  while (n >= days[m])
      m++;
  month_t month = m;
  int day = n - days[m - 1] + 1;
  return { year, month, day };
}
```

Ours

```
date to_date(int32_t N) { // Epoch is 0000-Mar-01

  // Century.
  uint32_t N_1 = 4 * N + 3;
  uint32_t C   = N_1 / 146097;
  uint32_t N_C = N_1 % 146097 / 4;

  // Year.
  uint32_t N_2 = 4 * N_C + 3;
  uint64_t P_2 = uint64_t(2939745) * N_2;
  uint32_t Z   = uint32_t(P_2 / 4294967296);
  uint32_t N_Y = uint32_t(P_2 % 4294967296) / 2939745 / 4;
  uint32_t Y   = 100 * C + Z;

  // Month and day.
  uint32_t N_3 = 2141 * N_Y + 197913;
  uint32_t M   = N_3 / 65536;
  uint32_t D   = N_3 % 65536 / 2141;

  // Map.
  uint32_t J   = N_Y >= 306;
  int32_t  Y_G = Y + J;
  uint32_t M_G = J ? M - 12 : M;
  uint32_t D_G = D + 1;

  return { Y_G, M_G, D_G };
}
```

Two solutions

.NET 6 (edited)

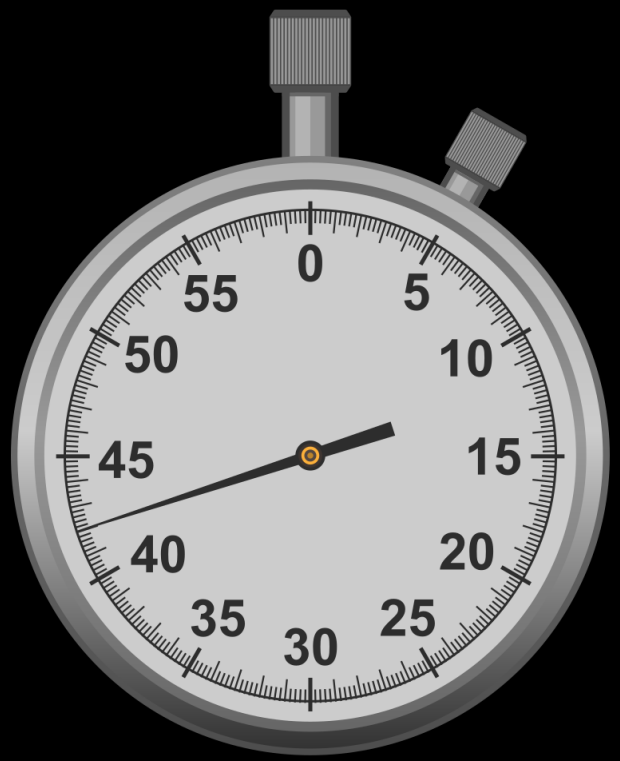
```
date to_date(int n) { // Epoch is 0001-Jan-01
  int s_daysToMonth365[] = {
    0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };
  int s_daysToMonth366[] = {
    0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 };

  int y400 = n / 146097;
  n -= y400 * 146097;
  int y100 = n / 36524;
  if (y100 == 4)
      y100 = 3;
  n -= y100 * 36524;
  int y4 = n / 1461;
  n -= y4 * 1461;
  int y1 = n / 365;
  if (y1 == 4)
      y1 = 3;
  int year = y400 * 400 + y100 * 100 + y4 * 4 + y1 + 1;
  n -= y1 * 365;
  bool leapYear = y1 == 3 && (y4 != 24 || y100 == 3);
  int* days = leapYear ? s_daysToMonth366 : s_daysToMonth365;
  int m = (n >> 5) + 1;
  while (n >= days[m])
      m++;
  month_t month = m;
  int day = n - days[m - 1] + 1;
  return { year, month, day };
}
```

Ours

```
date to_date(int32_t N) { // Epoch is 0000-Mar-01
  // Century.
  uint32_t N_1 = 4 * N + 3;
  uint32_t C   = N_1 / 146097;
  uint32_t N_C = N_1 % 146097 / 4;

  // Year.
  uint32_t N_2 = 4 * N_C + 3;
  uint64_t P_2 = uint64_t(2939745) * N_2;
  uint32_t Z   = uint32_t(P_2 / 4294967296);
  uint32_t N_Y = uint32_t(P_2 % 4294967296) / 2939745 / 4;
  uint32_t Y   = 100 * C + Z;

  // Month and day.
  uint32_t N_3 = 2141 * N_Y + 197913;
  uint32_t M   = N_3 / 65536;
  uint32_t D   = N_3 % 65536 / 2141;

  // Map.
  uint32_t J   = N_Y >= 306;
  int32_t  Y_G = Y + J;
  uint32_t M_G = J ? M - 12 : M;
  uint32_t D_G = D + 1;

  return { Y_G, M_G, D_G };
}
```

Performances

Performances




```
 5.0 Neri-Schneider
 7.3 Boost
 9.2 Baum
11.0 Hatcher
11.9 OpenJDK
12.2 Fliegel-Flandern
13.1 libc++
19.3 .NET 6
37.1 glibc
41.2 Reingold-Dershowitz
```

➜ gcc, Linux Kernel and .NET 7

➜ Android

➜ clang and msvc

Overview

HISTORY

Saturnalia, Macrobius

431 BCE

Julius Caesar
by
Peter Paul Rubens.

Pope Gregory XIII
by
Bartolomeo
Passarotti.

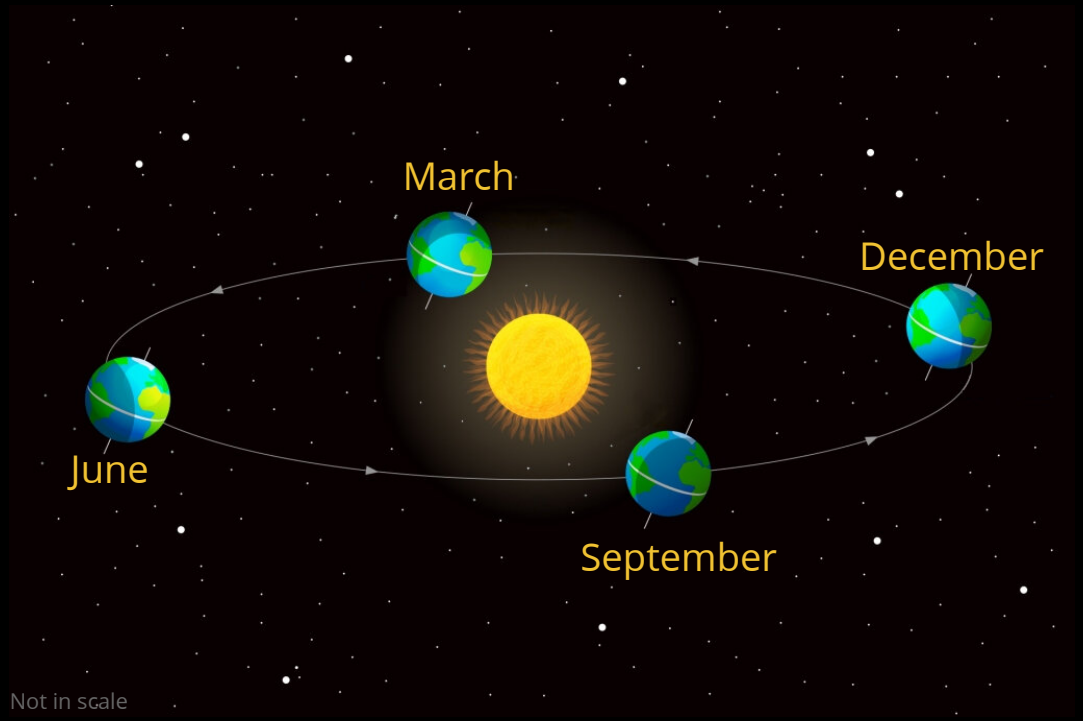
MATHS

Euclidean affine functions

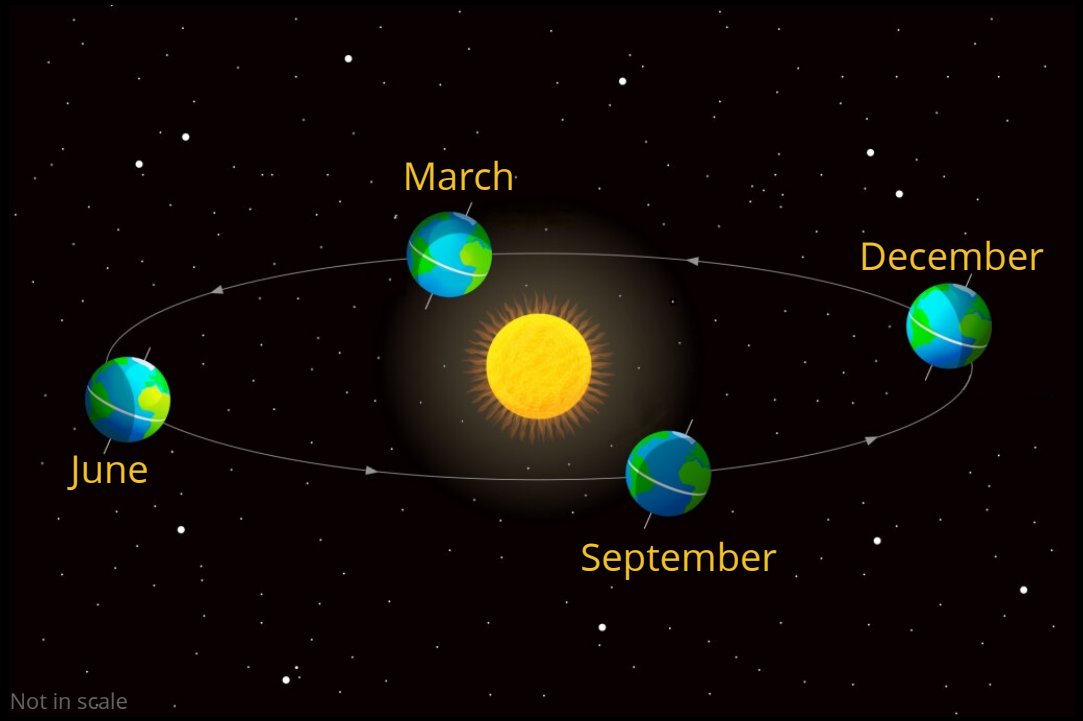
$$f(n) = (a \cdot n + b)/d$$

CODING

The origin of calendars

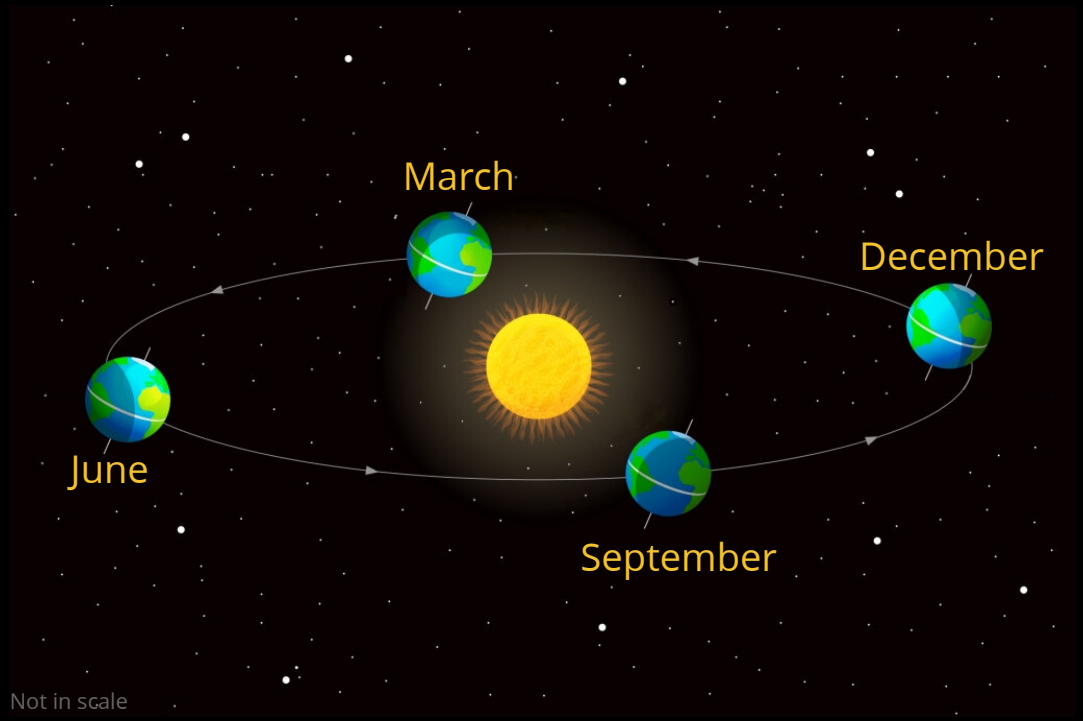


The origin of calendars



Earth movement	Effect
rotation	light and darkness ➜ **day**
revolution	seasons duration of the night equinoxes and solstices ➜ **year**

The origin of calendars



Not in scale

Earth movement	Effect
rotation	light and darkness ➜ **day**
revolution	seasons duration of the night equinoxes and solstices ➜ **year**

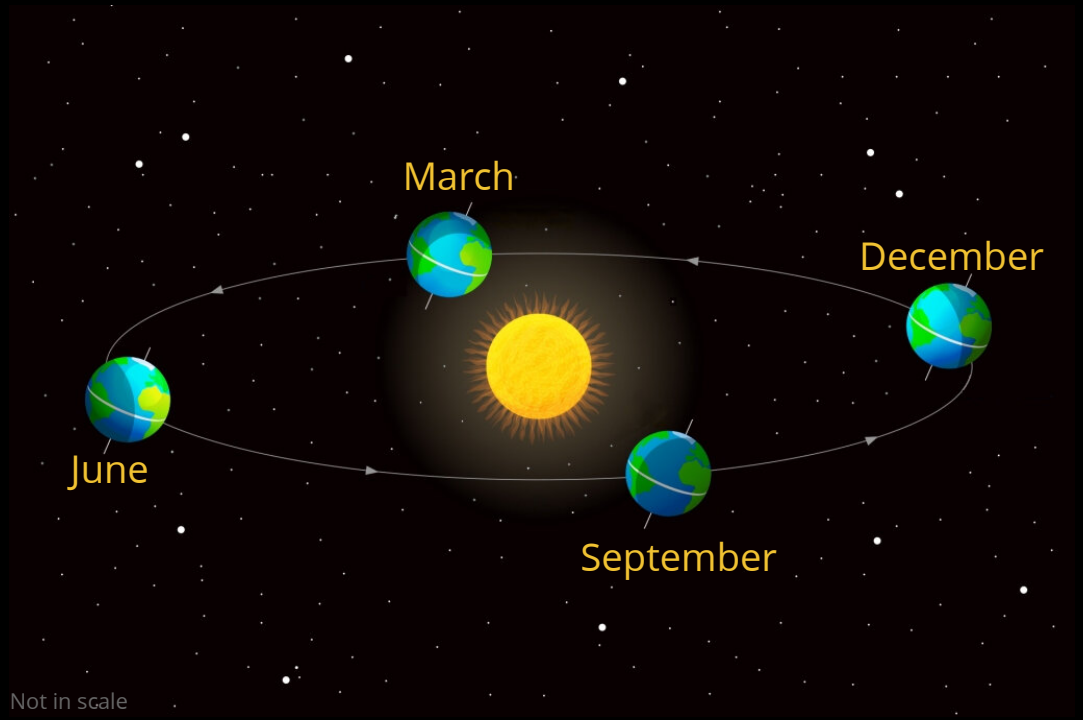
Stonehenge
England
≈ 2500 BCE

Mavratti, Public domain,
via Wikimedia Commons.

Wurdi Youang
Australia
≈ 11000 BCE

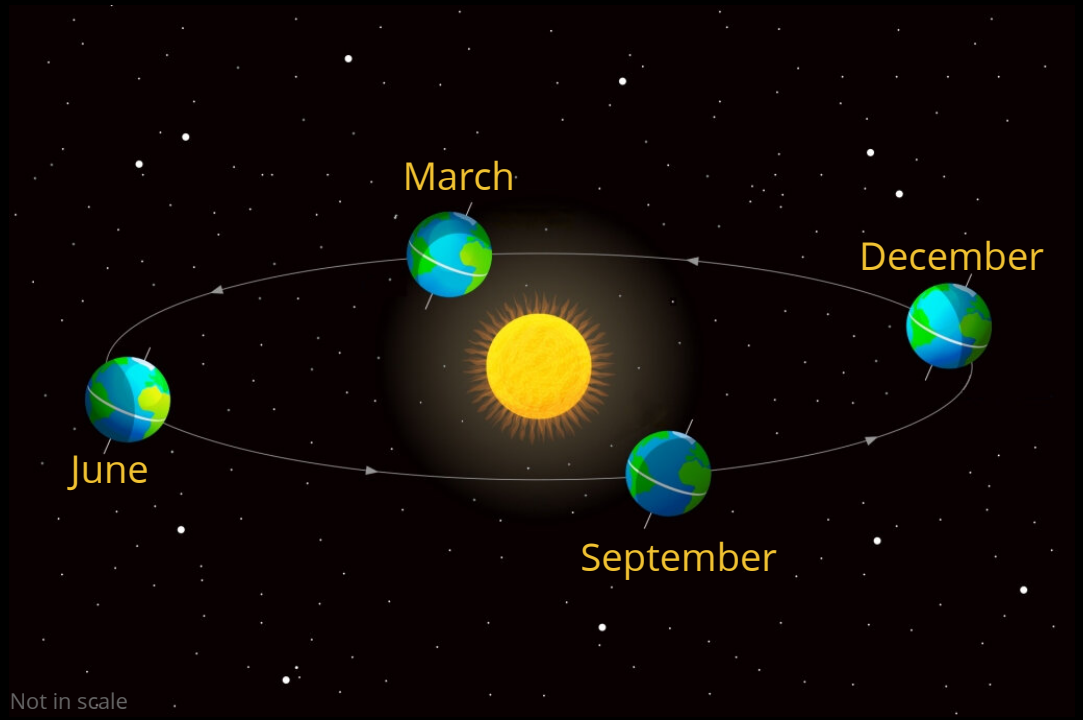
Marshall & Webb (1999)

The origin of calendars



Earth movement	Effect
rotation	light and darkness ➜ **day**
revolution	seasons duration of the night equinoxes and solstices ➜ **year**
Moon's revolution	Moon phases ➜ **month**

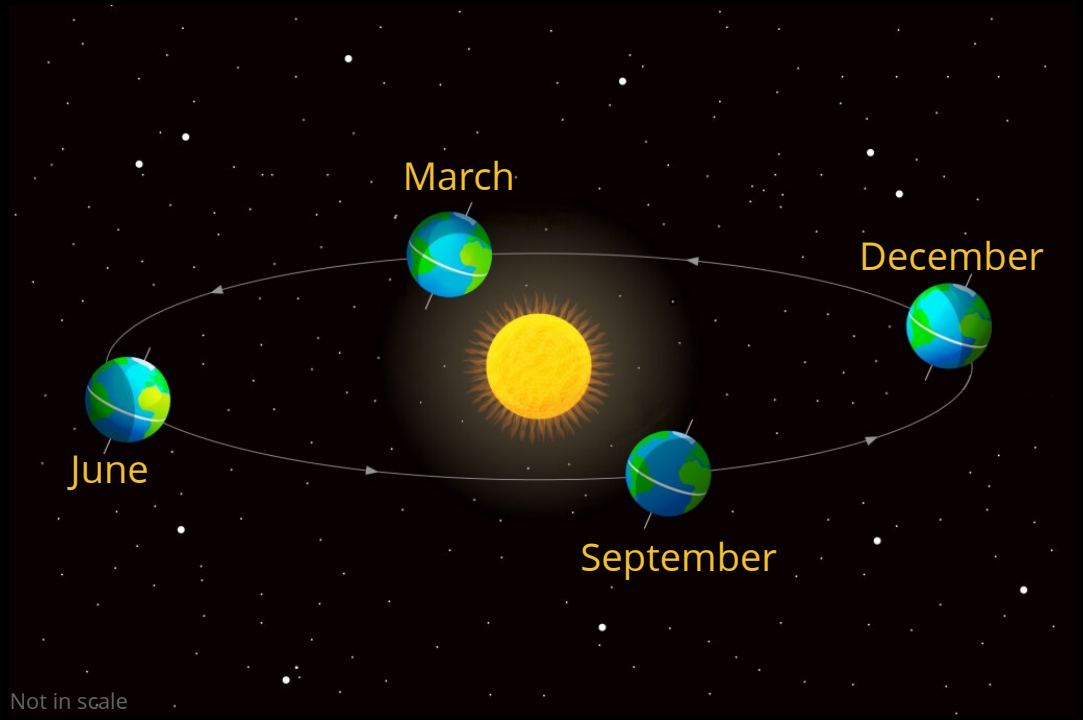
The origin of calendars



Earth movement	Effect
rotation	light and darkness ➜ **day**
revolution	seasons duration of the night equinoxes and solstices ➜ **year**
Moon's revolution	Moon phases ➜ **month**

Problem: find $n,\ m \in \mathbb{N}$ such that $n \cdot t_{\text{year}} = m \cdot t_{\text{day}}$, *i.e.*, $\frac{t_{\text{year}}}{t_{\text{day}}} = \frac{m}{n} \in \mathbb{Q}$.

The origin of calendars



Earth movement	Effect
rotation	light and darkness ➜ **day**
revolution	seasons duration of the night equinoxes and solstices ➜ **year**
Moon's revolution	Moon phases ➜ **month**

Problem: find $n,\ m \in \mathbb{N}$ such that $n \cdot t_{\text{year}} = m \cdot t_{\text{day}}$, *i.e.*, $\frac{t_{\text{year}}}{t_{\text{day}}} = \frac{m}{n} \in \mathbb{Q}$.

There's no solution since $\frac{t_{\text{year}}}{t_{\text{day}}} \notin \mathbb{Q}$. $\left(\frac{t_{\text{year}}}{t_{\text{day}}} \approx 365.2424.\right)$

Diana Preston, *Cleopatra and Antony*

“Caesar was not content with ruling the earth but wanted to rule the stars.”

Roman calendars - I

Martius	*Aprilis*	*Maius*
31	30	31
Junius	***Quintilis***	***Sextilis***
30	31	30
September	***October***	***November***
30	31	30
December	Winter	
30	?	

Roman calendars - I

Martius	*Aprilis*	*Maius*
31	30	31
Junius	***Quintilis***	***Sextilis***
30	31	30
September	***October***	***November***
30	31	30
December	Winter	
30	?	

- From the 5th, months had ordinal names.
- 10 months, 304 days.
- Duration of the year decided by the *Pontifices*.

Roman calendars - II

Januarius	*Martius*	*Aprilis*
29	31	29
Maius	***Junius***	***Quintilis***
31	29	31
Sextilis	***September***	***October***
29	29	31
November	***December***	***Februarius***
29	29	28,50,51

Roman calendars - II

Januarius	*Martius*	*Aprilis*
29	31	29
Maius	***Junius***	***Quintilis***
31	29	31
Sextilis	***September***	***October***
29	29	31
November	***December***	***Februarius***
29	29	28,50,51

- 355, 377 or 378 days.
- Intercalation was too complex and ignored.
- Each year length still decided by the *Pontifices.*

Roman calendars - III

Januarius	*Februarius*	*Martius*
29	28,50,51	31
Aprilis	***Maius***	***Junius***
29	31	29
Quintilis	***Sextilis***	***September***
31	29	29
October	***November***	***December***
31	29	29

Roman calendars - III

Januarius	*Februarius*	*Martius*
29	28,50,51	31
Aprilis	***Maius***	***Junius***
29	31	29
Quintilis	***Sextilis***	***September***
31	29	29
October	***November***	***December***
31	29	29

- Year length still decided by the *Pontifices* on increasingly political basis.

Roman calendars - IV

Januarius **31**	***Februarius*** **28**,29	***Martius*** **31**
Aprilis **30**	***Maius*** **31**	***Junius*** **30**
Quintilis **31**	***Sextilis*** **31**	***September*** **30**
October **31**	***November*** **30**	***December*** **31**

Roman calendars - IV

Januarius **31**	***Februarius*** **28**,29	***Martius*** **31**
Aprilis **30**	***Maius*** **31**	***Junius*** **30**
Quintilis **31**	***Sextilis*** **31**	***September*** **30**
October **31**	***November*** **30**	***December*** **31**

- Intercalation: every 4th year is a leap year.

$$\frac{3 \cdot 365 + 1 \cdot 366}{4} = 365.25$$

365.2424

Roman calendars - IV

Januarius **31**	***Februarius*** **28,29**	***Martius*** **31**
Aprilis **30**	***Maius*** **31**	***Junius*** **30**
Quintilis **31**	***Sextilis*** **31**	***September*** **30**
October **31**	***November*** **30**	***December*** **31**

- Intercalation: every 4th year is a leap year.

$$\frac{3 \cdot 365 + 1 \cdot 366}{4} = 365.25$$

365.2424

- The *Pontifices* misunderstood it for every 3rd year. 🤦

Roman calendars - V

Januarius	*Februarius*	*Martius*
31	28,29	31
Aprilis	***Maius***	***Junius***
30	31	30
Julius	***Augustus***	***September***
31	31	30
October	***November***	***December***
31	30	31

Roman calendars - V

Januarius **31**	***Februarius*** **28,29**	***Martius*** **31**
Aprilis **30**	***Maius*** **31**	***Junius*** **30**
Julius **31**	***Augustus*** **31**	***September*** **30**
October **31**	***November*** **30**	***December*** **31**

- Suspended leap years until alignment.
- Every 4th year is a leap year.
- This is the so called Julian calendar.

Gregorian calendar

Januarius **31**	***Februarius*** **28,29**	***Martius*** **31**
Aprilis **30**	***Maius*** **31**	***Junius*** **30**
Julius **31**	***Augustus*** **31**	***September*** **30**
October **31**	***November*** **30**	***December*** **31**

Gregorian calendar

Januarius **31**	***Februarius*** **28,29**	***Martius*** **31**
Aprilis **30**	***Maius*** **31**	***Junius*** **30**
Julius **31**	***Augustus*** **31**	***September*** **30**
October **31**	***November*** **30**	***December*** **31**

- Intercalation:

365.2425

Gregorian calendar

Januarius	*Februarius*	*Martius*
31	28,29	31
Aprilis	***Maius***	***Junius***
30	31	30
Julius	***Augustus***	***September***
31	31	30
October	***November***	***December***
31	30	31

- Intercalation:

$$365.2425 = 365.25 - 0.01 + 0.0025$$
$$= 365.25 - \frac{1}{100} + \frac{1}{400}$$

365.2424

Gregorian calendar

Januarius **31**	***Februarius*** **28,29**	***Martius*** **31**
Aprilis **30**	***Maius*** **31**	***Junius*** **30**
Julius **31**	***Augustus*** **31**	***September*** **30**
October **31**	***November*** **30**	***December*** **31**

- Intercalation:

$$365.2425 = 365.25 - 0.01 + 0.0025$$
$$= 365.25 - \frac{1}{100} + \frac{1}{400}$$

365.2424

- A leap year is one that is multiple of 4 **except those that are multiple of 100 but not 400.**

Adoptions

Julian calendar

- 46 BCE had 445 days. (*Annus confusionis ultimus*.)

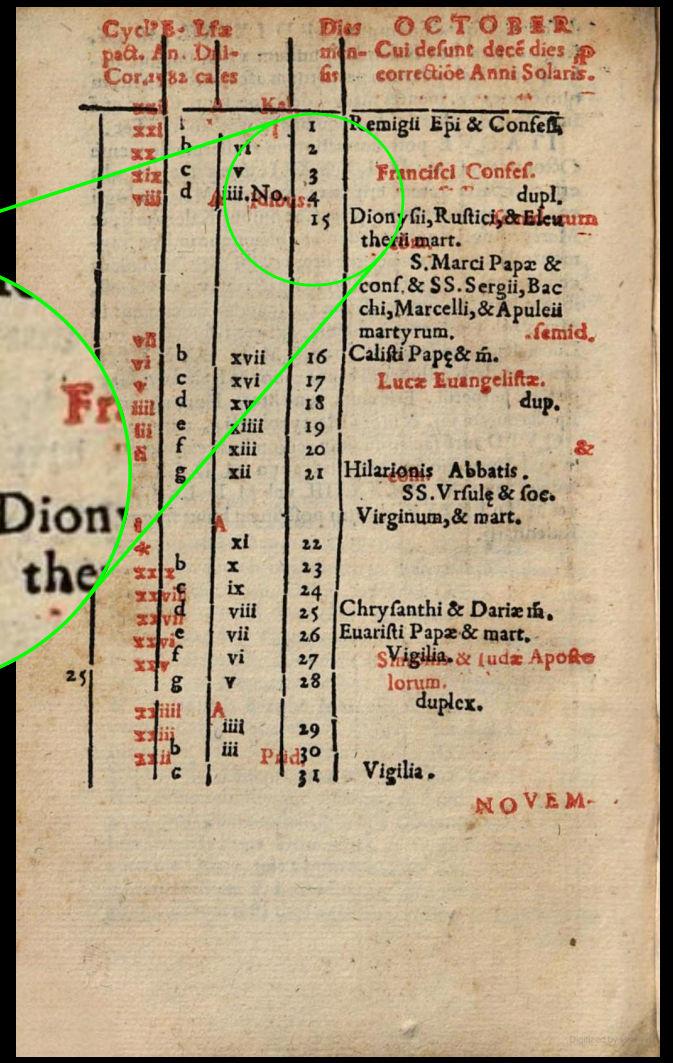
Adoptions

Julian calendar

- 46 BCE had 445 days. (*Annus confusionis ultimus*.)

Gregorian calendar

- Some catholic countries adopted it in 1582:
  4 October was followed by 15.



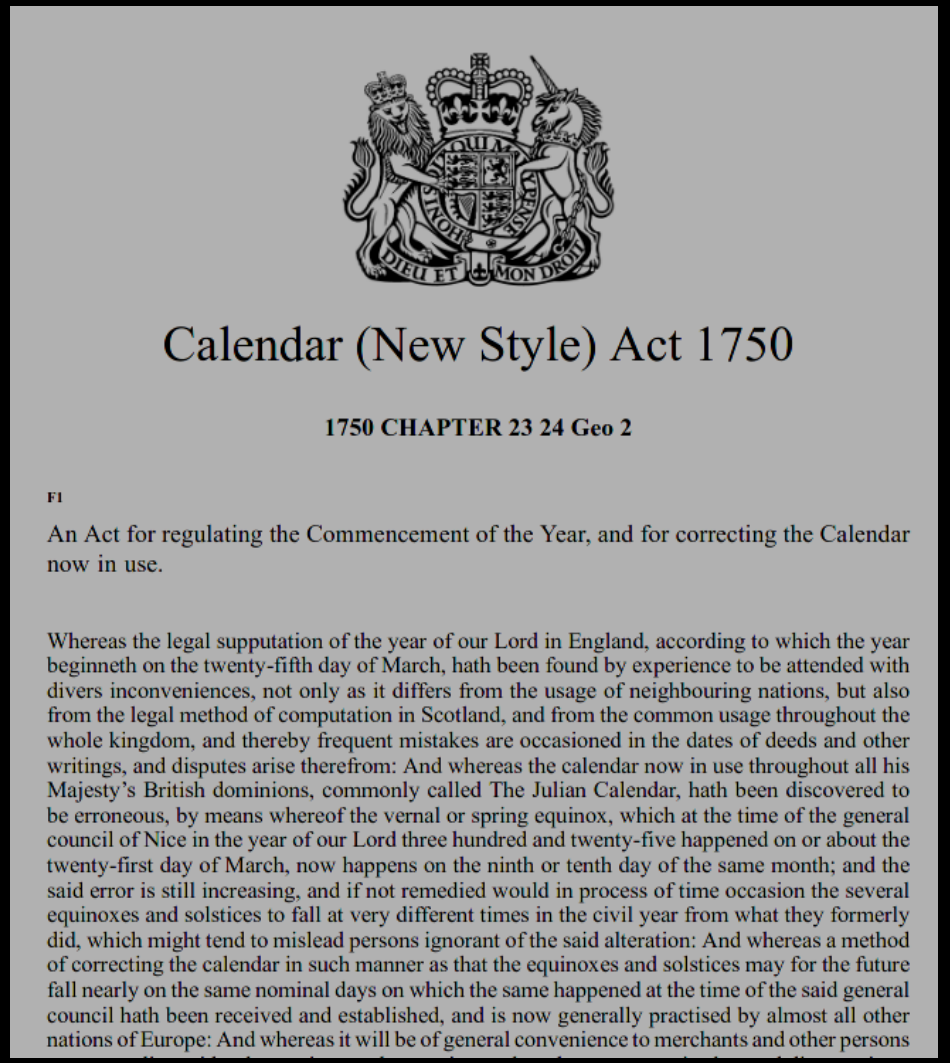
Adoptions

Julian calendar

- 46 BCE had 445 days. (*Annus confusionis ultimus*.)

Gregorian calendar

- Some catholic countries adopted it in 1582:
  4 October was followed by 15.
- Great Britain adopted it in 1752:
  2 September was followed by 14.



Calendar (New Style) Act 1750

1750 CHAPTER 23 24 Geo 2

F1

An Act for regulating the Commencement of the Year, and for correcting the Calendar now in use.

Whereas the legal supputation of the year of our Lord in England, according to which the year beginneth on the twenty-fifth day of March, hath been found by experience to be attended with divers inconveniences, not only as it differs from the usage of neighbouring nations, but also from the legal method of computation in Scotland, and from the common usage throughout the whole kingdom, and thereby frequent mistakes are occasioned in the dates of deeds and other writings, and disputes arise therefrom: And whereas the calendar now in use throughout all his Majesty's British dominions, commonly called The Julian Calendar, hath been discovered to be erroneous, by means whereof the vernal or spring equinox, which at the time of the general council of Nice in the year of our Lord three hundred and twenty-five happened on or about the twenty-first day of March, now happens on the ninth or tenth day of the same month; and the said error is still increasing, and if not remedied would in process of time occasion the several equinoxes and solstices to fall at very different times in the civil year from what they formerly did, which might tend to mislead persons ignorant of the said alteration: And whereas a method of correcting the calendar in such manner as that the equinoxes and solstices may for the future fall nearly on the same nominal days on which the same happened at the time of the said general council hath been received and established, and is now generally practised by almost all other nations of Europe: And whereas it will be of general convenience to merchants and other persons

Adoptions

Julian calendar

- 46 BCE had 445 days. (*Annus confusionis ultimus*.)

Gregorian calendar

- Some catholic countries adopted it in 1582:
  4 October was followed by 15.
- Great Britain adopted it in 1752:
  2 September was followed by 14.

```
$ cal 9 1752
    September 1752
Mo Tu We Th Fr Sa Su
     1  2 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

gcc's manual

“Programmers are notoriously bad at predicting how their programs actually perform.”

Is Y a leap year?

Classical

```
return Y % 4 == 0 && (Y % 100 != 0 || Y % 400 == 0);
```

Is Y a leap year?

Classical

```
if (Y % 4 != 0)
  return false;

if (Y % 100 != 0)
  return true;

return Y % 400 == 0;
```

Is Y a leap year?

Classical

```
if (Y % 4 != 0)                    75%
  return false;

if (Y % 100 != 0)                  25%
  return true;

return Y % 400 == 0;
```

- Single divisibility check in 75% of the cases.

Is Y a leap year?

Classical

```
if (Y % 4 != 0)                    75%
  return false;
```

```
if (Y % 100 != 0)                  25%
  return true;

return Y % 400 == 0;
```

- Single divisibility check in 75% of the cases.

Ours

```
if (Y % 100 != 0)
  return Y % 4 == 0;

return Y % 400 == 0;
```

- Always does two divisibility checks.

Is Y a leap year?

Classical

```
if (Y % 4 != 0)                    75%
  return false;
```

```
if (Y % 100 != 0)                  25%
  return true;

return Y % 400 == 0;
```

- Single divisibility check in 75% of the cases.
- 3x slower!

Ours

```
if (Y % 100 != 0)
  return Y % 4 == 0;

return Y % 400 == 0;
```

- Always does two divisibility checks.
- 3x faster!

Is Y a leap year?

Classical

```
if (Y % 4 != 0)                    75%
  return false;
```

```
if (Y % 100 != 0)                  25%
  return true;

return Y % 400 == 0;
```

- Single divisibility check in 75% of the cases.
- 3x slower!
- Higher entropy weakens the branch predictor.

Ours

```
if (Y % 100 != 0)                  99%
  return Y % 4 == 0;
```

```
return Y % 400 == 0;               1%
```

- Always does two divisibility checks.
- 3x faster!
- Lower entropy helps the branch predictor.

Is Y a leap year?

Classical

```
if (Y % 4 != 0)                    75%
  return false;
```

```
if (Y % 100 != 0)                  25%
  return true;

return Y % 400 == 0;
```

- Single divisibility check in 75% of the cases.
- 3x slower!
- Higher entropy weakens the branch predictor.

Ours

```
if (Y % 100 != 0)                  99%
  return Y % 4 == 0;
```

```
return Y % 16 == 0;                1%
```

- Always does two divisibility checks.
- 3x faster!
- Lower entropy helps the branch predictor.

Is Y a leap year?

Classical

```
if (Y % 4 != 0)                    75%
  return false;
```

```
if (Y % 100 != 0)                  25%
  return true;

return Y % 400 == 0;
```

- Single divisibility check in 75% of the cases.
- 3x slower!
- Higher entropy weakens the branch predictor.

Ours

```
if (Y % 100 != 0)                  99%
  return Y % 4 == 0;
```

```
return Y % 16 == 0;                1%
```

```
int d = Y % 100 != 0 ? 4 : 16;
return (Y % d) == 0;
```

- Always does two divisibility checks.
- 3x faster!
- Lower entropy helps the branch predictor.

Is Y a leap year?

Classical

```
if (Y % 4 != 0)                    75%
  return false;
```

```
if (Y % 100 != 0)                  25%
  return true;

return Y % 400 == 0;
```

- Single divisibility check in 75% of the cases.
- 3x slower!
- Higher entropy weakens the branch predictor.

Ours

```
if (Y % 100 != 0)                  99%
  return Y % 4 == 0;
```

```
return Y % 16 == 0;                1%
```

```
int d = Y % 100 != 0 ? 4 : 16;
return (Y & (d - 1)) == 0;
```

- Always does two divisibility checks.
- 3x faster!
- Lower entropy helps the branch predictor.

Last day of month

Classical

```
if (M == 2)
  return is_leap(Y) ? 29 : 28;

unsigned last[] = {
  31, 28, 31, 30, 31, 30,
  31, 31, 30, 31, 30, 31
};

return last[M - 1];
```

Last day of month

Classical

```
if (M == 2)
  return is_leap(Y) ? 29 : 28;

unsigned last[] = {
  31, 28, 31, 30, 31, 30,
  31, 31, 30, 31, 30, 31
};

return last[M - 1];
```

Last day of month

Classical

```
if (M == 2)
  return is_leap(Y) ? 29 : 28;

unsigned last[] = {
  31, 28, 31, 30, 31, 30,
  31, 31, 30, 31, 30, 31
};

return last[M - 1];
```

Ours

```
if (M == 2)
  return is_leap(Y) ? 29 : 28;

return 30 | (M ^ (M >> 3));
```

Last day of month

Classical

```
if (M == 2)
  return is_leap(Y) ? 29 : 28;

unsigned last[] = {
  31, 28, 31, 30, 31, 30,
  31, 31, 30, 31, 30, 31
};

return last[M - 1];
```

Ours

```
if (M == 2)
  return is_leap(Y) ? 29 : 28;

return 30 | (M ^ (M >> 3));
```

Last day of month

Classical

```
if (M == 2)
  return is_leap(Y) ? 29 : 28;

unsigned last[] = {
  31, 28, 31, 30, 31, 30,
  31, 31, 30, 31, 30, 31
};

return last[M - 1];
```

Ours

```
if (M == 2)
  return is_leap(Y) ? 29 : 28;

return 30 | (M ^ (M >> 3));
```

Alternative

```
if (M == 2)
  return is_leap(Y) ? 29 : 28;

return 30 | (9 * M / 8);
```

Charles Caleb Colton

“The study of mathematics, like the Nile, begins in minuteness but ends in magnificence.”

Egyptian calendar - I

Years have 365 days.

Date N days after 1$^{\text{st}}$ day of year 0:

$$y(N) = \text{year} = Y$$

$$y^{\circ}(N) = \text{day of the year} = N_Y$$

$$y^{*}(Y) = \#\text{days before year } Y$$

Egyptian calendar - I

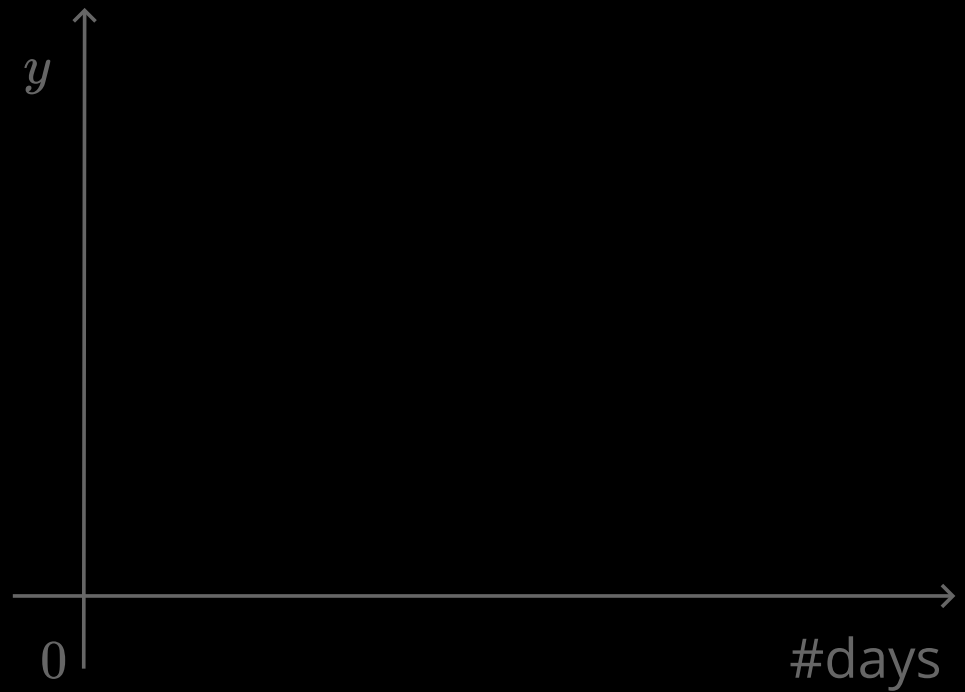
Years have 365 days.

Date N days after 1st day of year 0:

$$y(N) = \text{year} = Y$$
$$= N/365$$

$$y^\circ(N) = \text{day of the year} = N_Y$$
$$= N\%365$$

$$y^*(Y) = \#\text{days before year } Y$$
$$= 365 \cdot M$$



Egyptian calendar - I

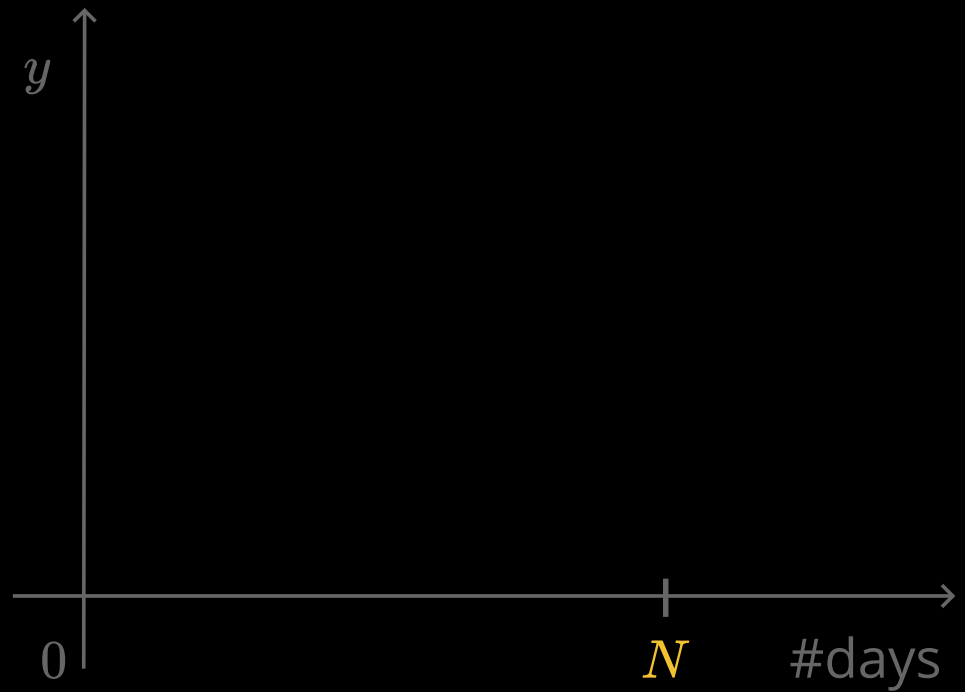
Years have 365 days.

Date N days after 1st day of year 0:

$$y(N) = \text{year} = Y$$
$$= N/365$$

$$y^\circ(N) = \text{day of the year} = N_Y$$
$$= N\%365$$

$$y^*(Y) = \#\text{days before year } Y$$
$$= 365 \cdot M$$



Egyptian calendar - I

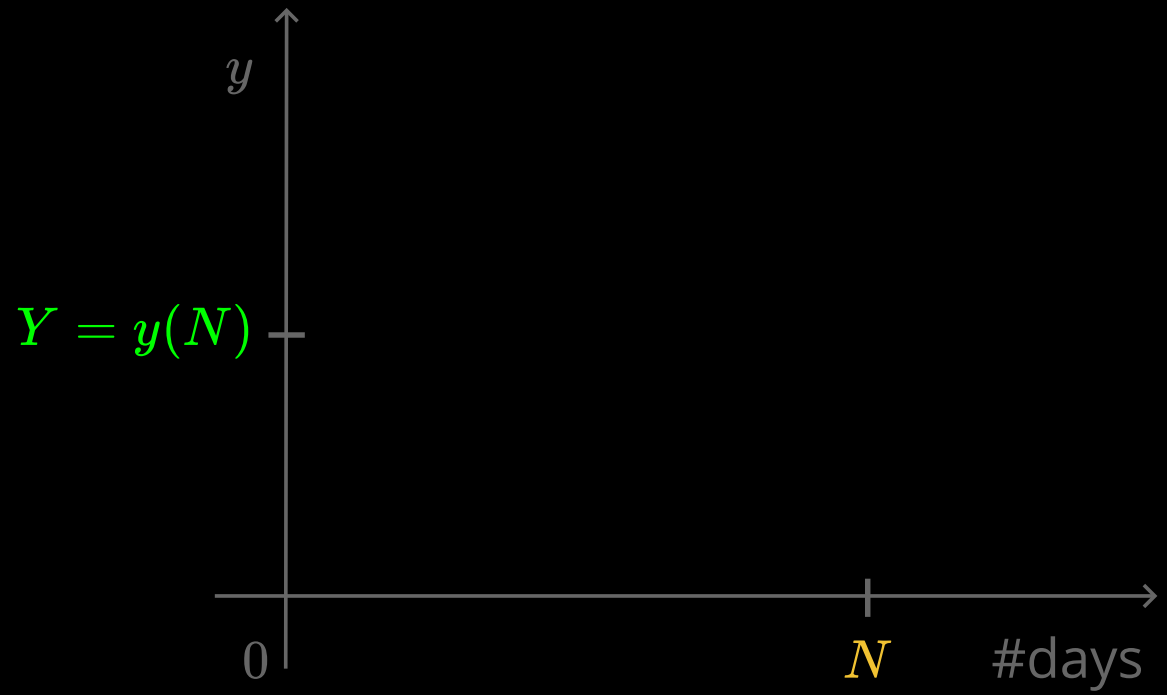
Years have 365 days.

Date N days after 1st day of year 0:

$$y(N) = \text{year} = Y$$
$$= N/365$$

$$y^\circ(N) = \text{day of the year} = N_Y$$
$$= N\%365$$

$$y^*(Y) = \#\text{days before year } Y$$
$$= 365 \cdot M$$



Egyptian calendar - I

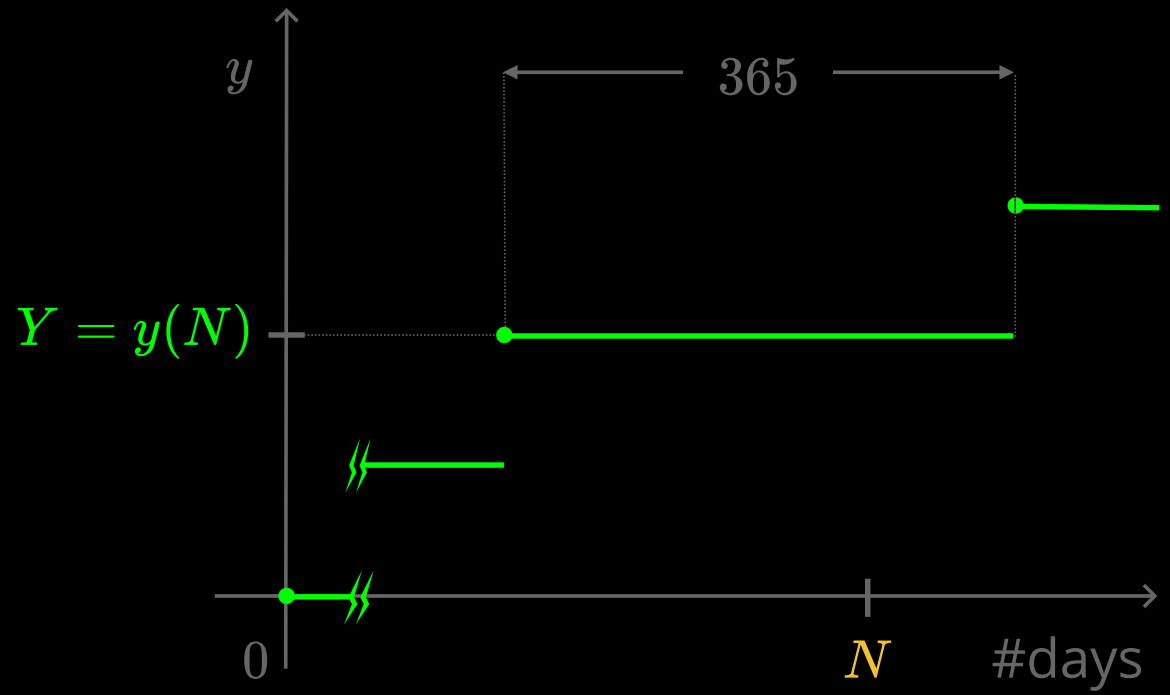
Years have 365 days.

Date N days after 1st day of year 0:

$$y(N) = \text{year} = Y$$
$$= N/365$$

$$y^\circ(N) = \text{day of the year} = N_Y$$
$$= N\%365$$

$$y^*(Y) = \#\text{days before year } Y$$
$$= 365 \cdot M$$



Egyptian calendar - I

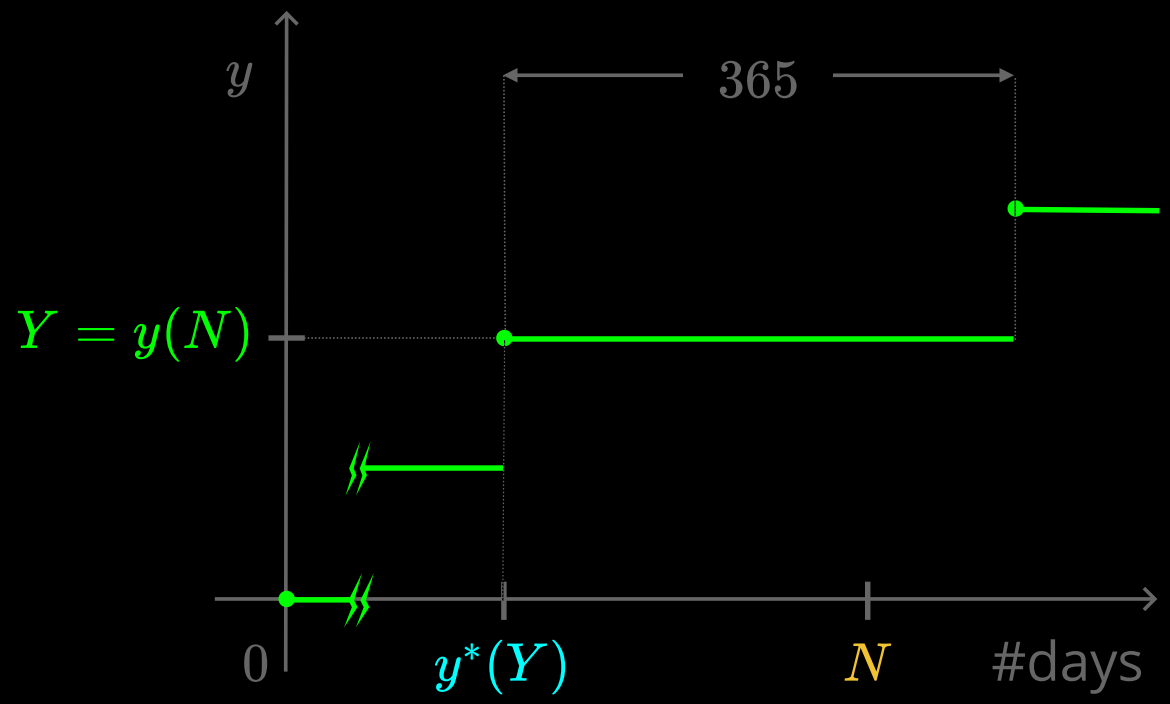
Years have 365 days.

Date N days after 1st day of year 0:

$$y(N) = \text{year} = Y$$
$$= N/365$$

$$y^\circ(N) = \text{day of the year} = N_Y$$
$$= N \% 365$$

$$y^*(Y) = \#\text{days before year } Y$$
$$= 365 \cdot M$$



Egyptian calendar - I

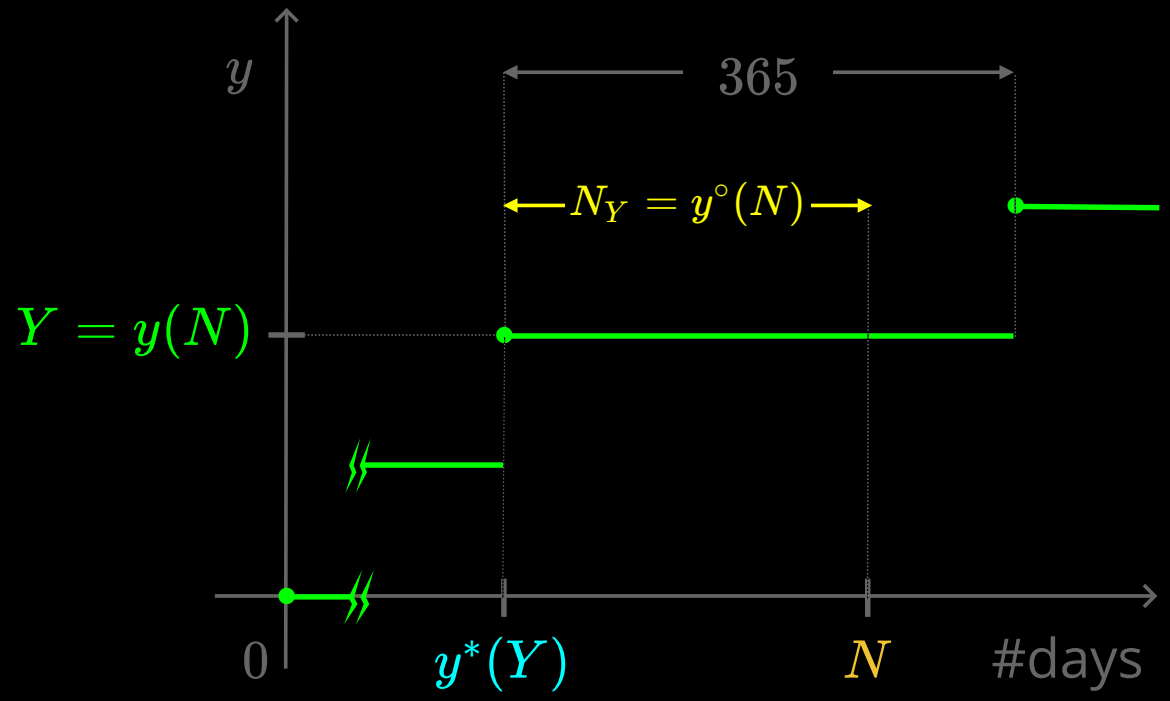
Years have 365 days.

Date N days after 1st day of year 0:

$$y(N) = \text{year} = Y$$
$$= N/365$$

$$y^\circ(N) = \text{day of the year} = N_Y$$
$$= N\%365$$

$$y^*(Y) = \#\text{days before year } Y$$
$$= 365 \cdot M$$



Egyptian calendar - I

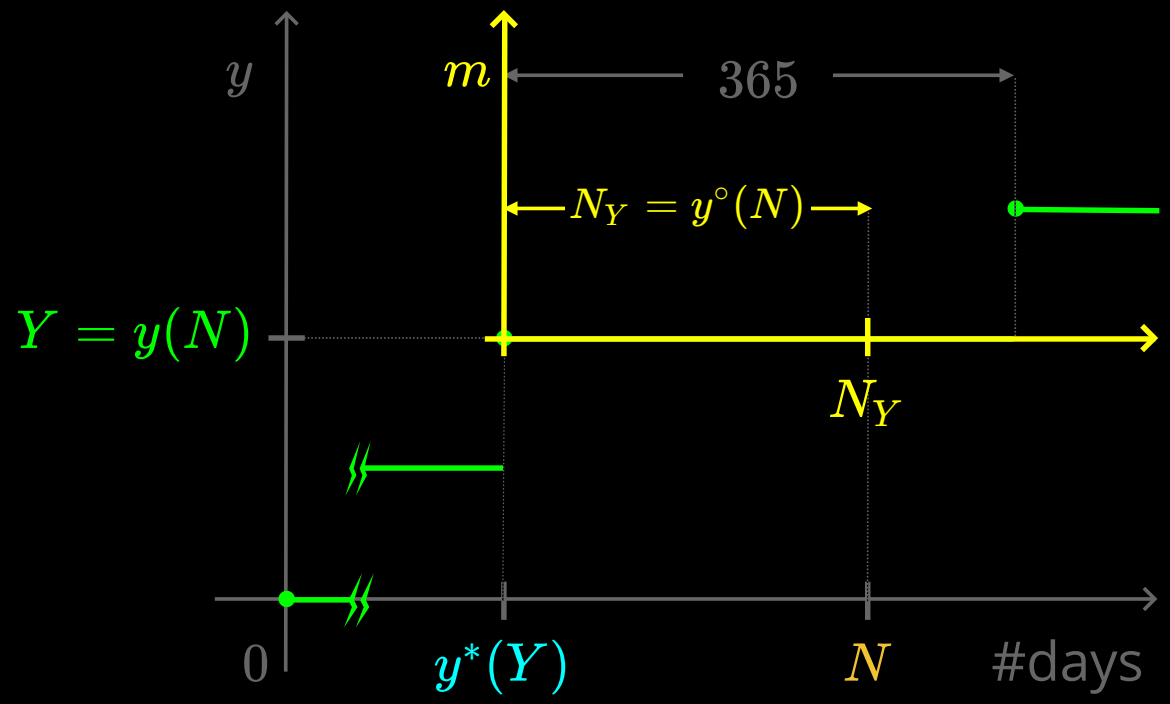
Years have 365 days.

Date N days after 1st day of year 0:

$$y(N) = \text{year} = Y$$
$$= N/365$$

$$y^\circ(N) = \text{day of the year} = N_Y$$
$$= N\%365$$

$$y^*(Y) = \#\text{days before year } Y$$
$$= 365 \cdot M$$



Egyptian calendar - II

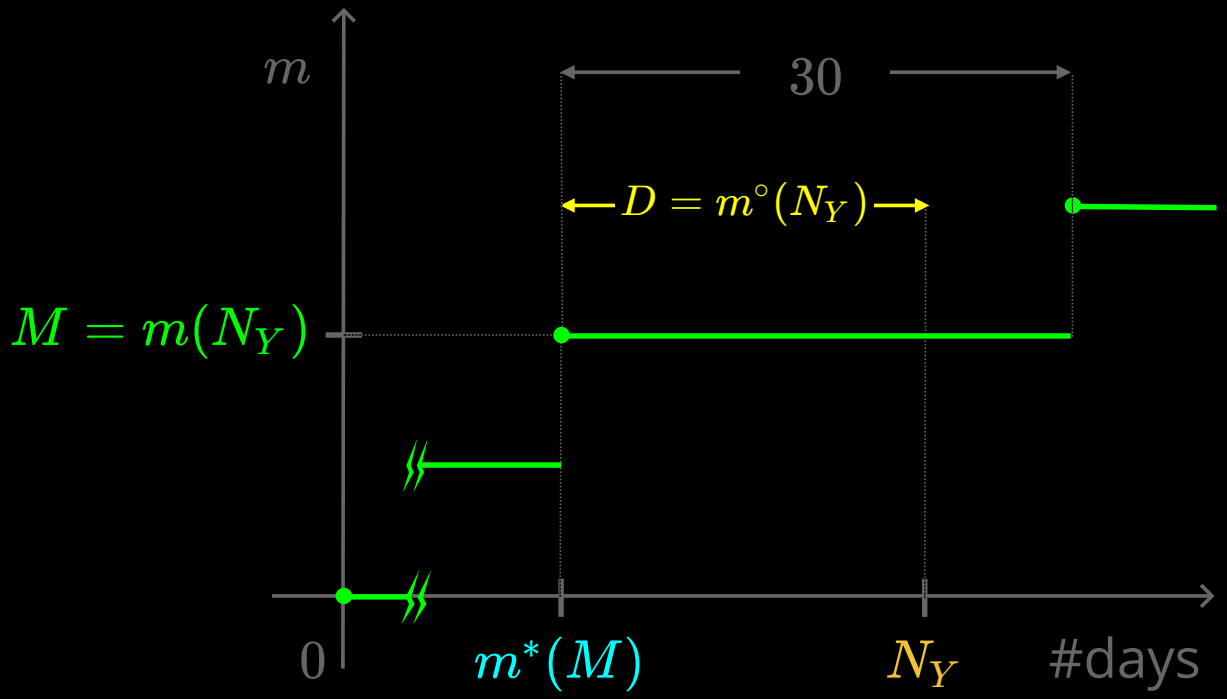
12 months of 30 days and 1 "month" of 5 days.

Date N_Y days after 1st day of the year:

$m(N_Y) =$ month $= M$

$m^\circ(N_Y) =$ day (of the month) $= D$

$m^*(M) =$ #days before month M



Egyptian calendar - II

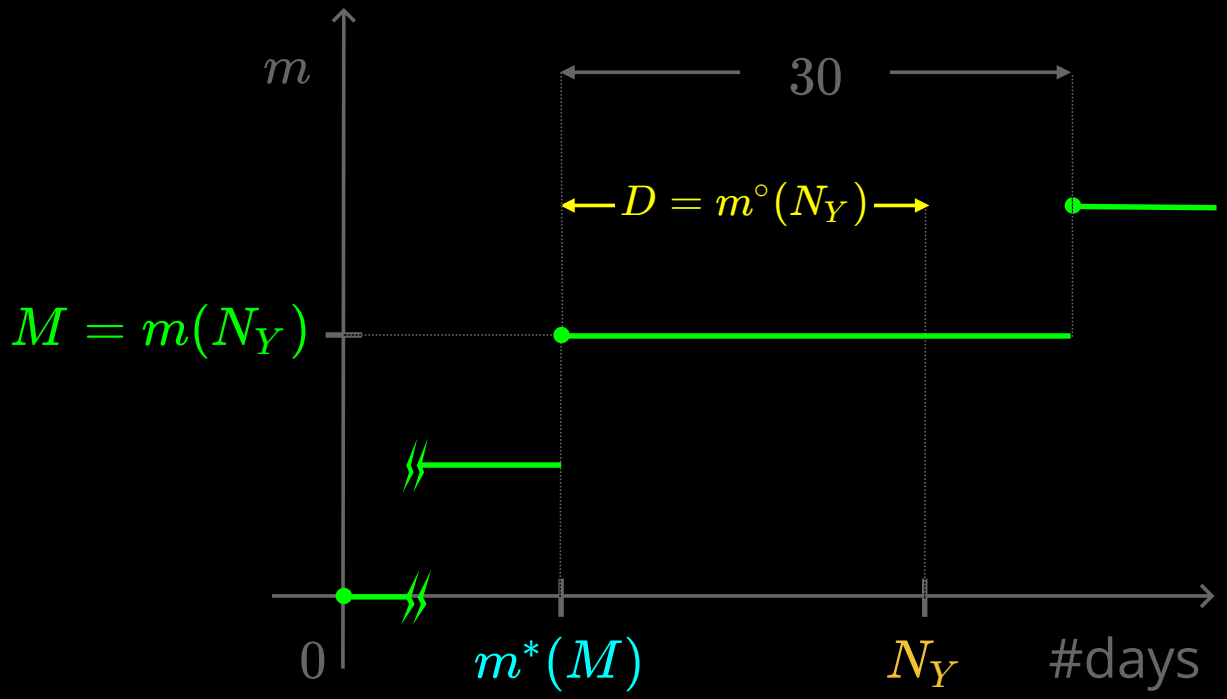
12 months of 30 days and 1 "month" of 5 days.

Date N_Y days after 1st day of the year:

$$m(N_Y) = \text{month} = M$$
$$= N_Y / 30$$

$$m^\circ(N_Y) = \text{day (of the month)} = D$$
$$= N_Y \,\%\, 30$$

$$m^*(M) = \#\text{days before month } M$$
$$= 30 \cdot M$$



Egyptian calendar - II

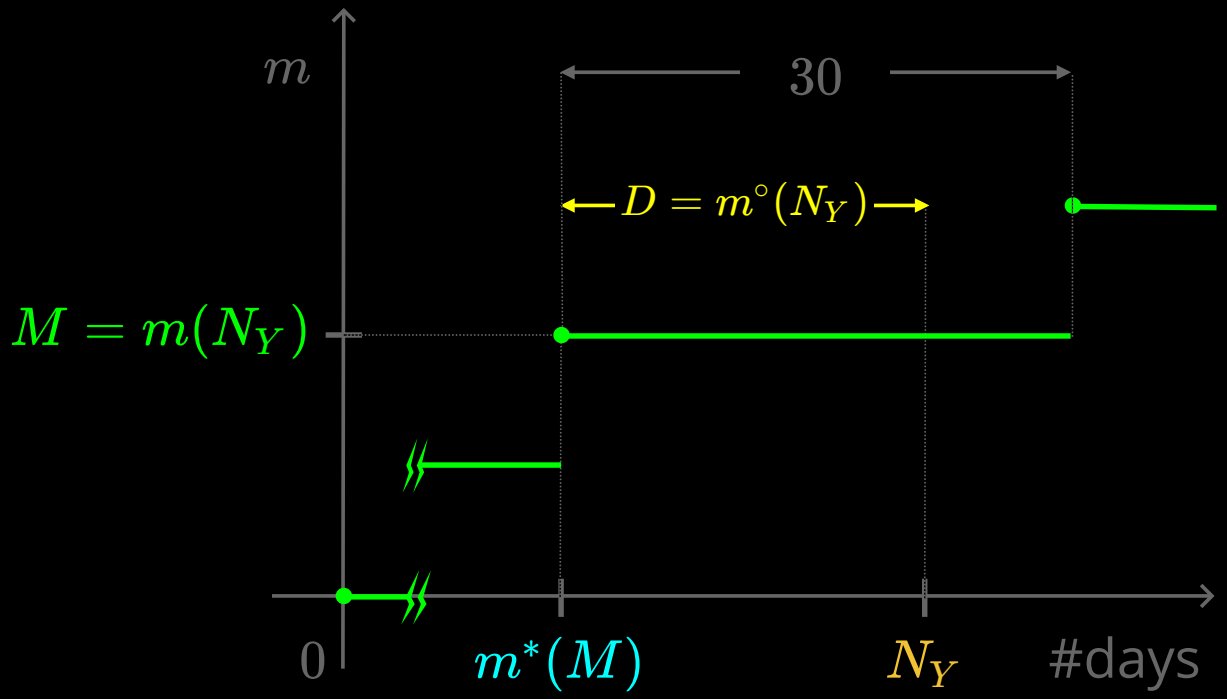
12 months of 30 days and 1 "month" of 5 days.

Date N_Y days after 1st day of the year:

$$m(N_Y) = \text{month} = M$$
$$= N_Y / 30$$

$$m^\circ(N_Y) = \text{day (of the month)} = D$$
$$= N_Y \,\%\, 30$$

$$m^*(M) = \#\text{days before month } M$$
$$= 30 \cdot M$$



Formulas are also correct for the last "month".

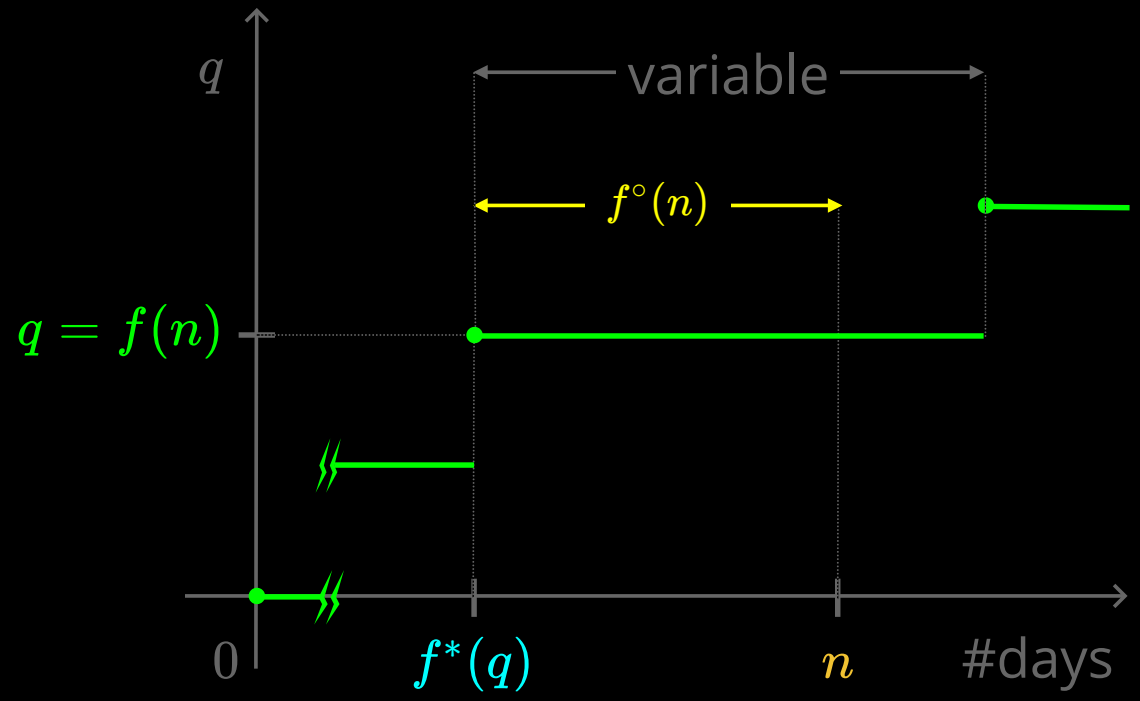
Calendrical functions

Date n days after the 1st day of the 1st period:

$f(n) =$ period $= q$

$f^\circ(n) =$ day of the period

$f^*(q) =$ #days before period q



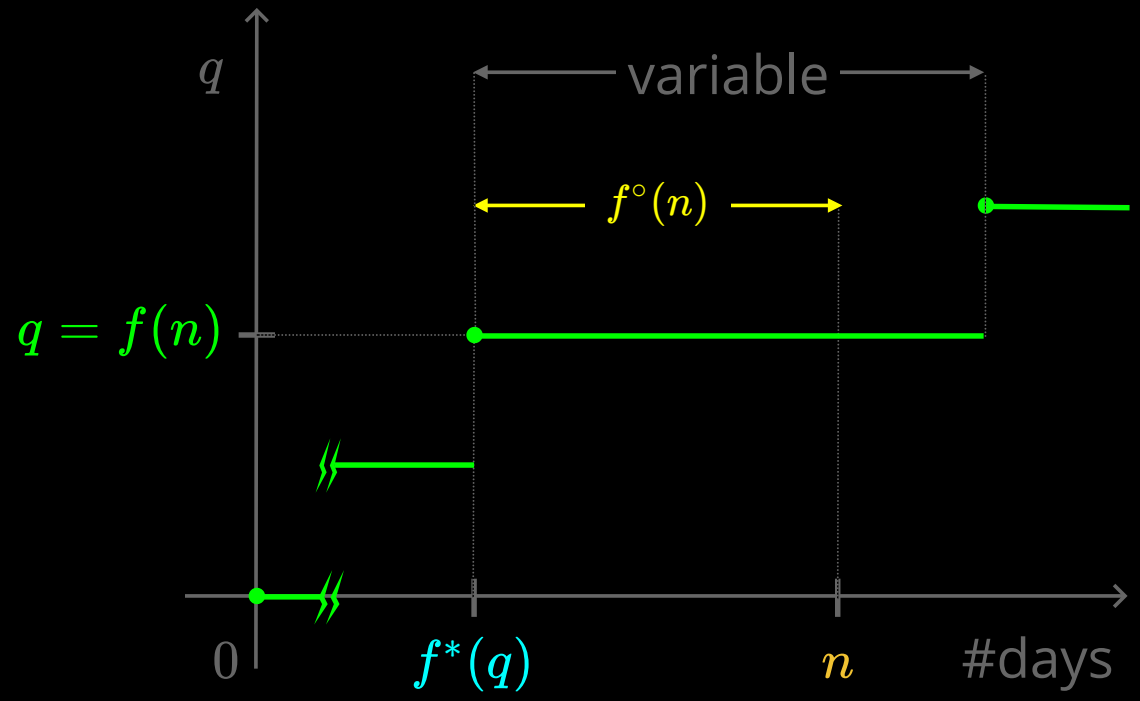
Calendrical functions

Date n days after the 1st day of the 1st period:

$f(n) =$ period $= q$

$f^\circ(n) =$ day of the period

$f^*(q) =$ #days before period q

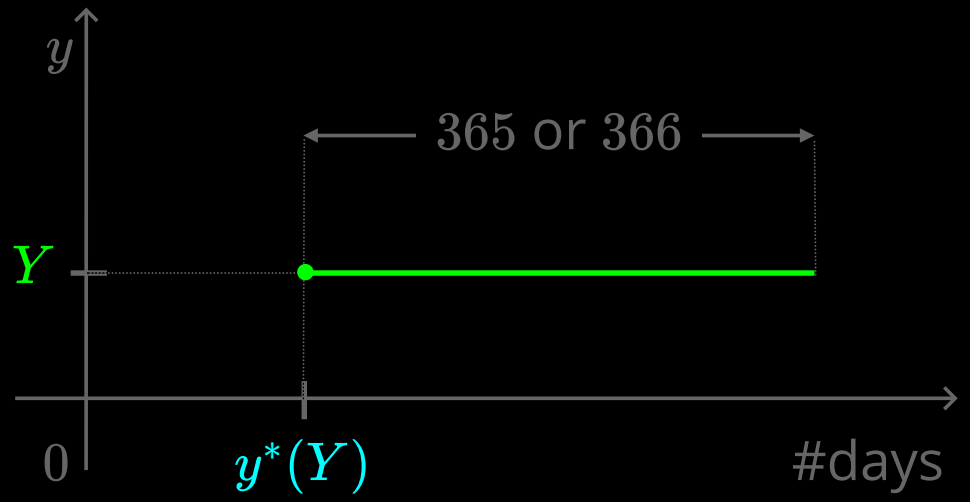


$$f^\circ(n) = n - f^*(f(n))$$

$$f^*(q) = \min\{\, n \,;\; f(n) = q \,\}$$

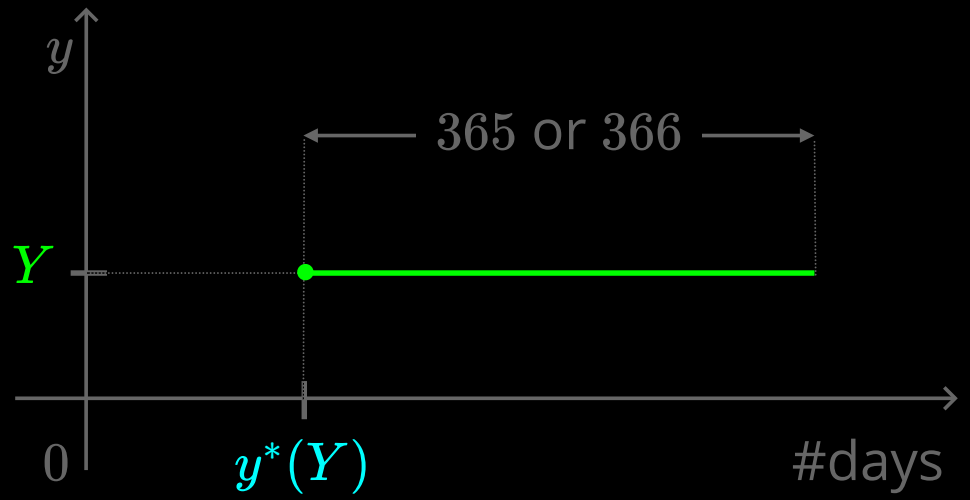
Julian calendar

Year Y is a leap year $\iff Y\%4 = 0$.



Julian calendar

Year Y is a leap year $\iff Y\%4 = 0$.

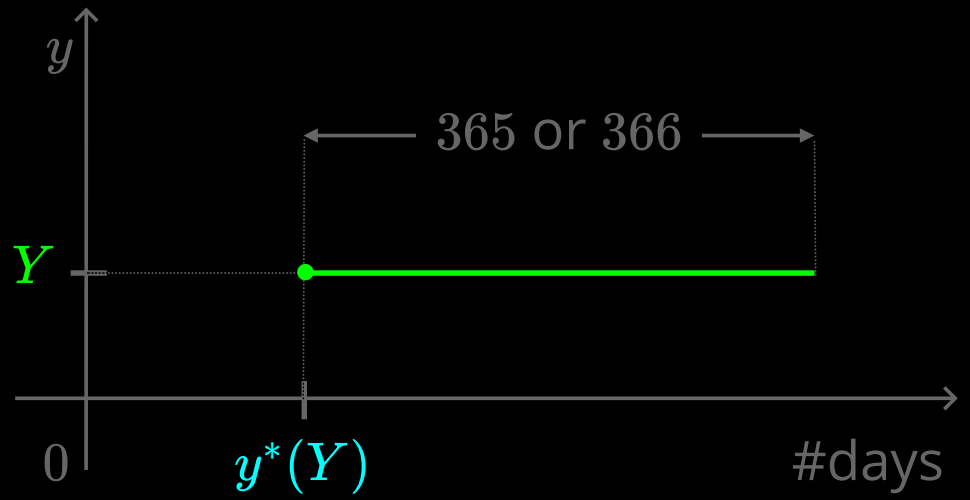


Reasonable attempt:

$$y^*(Y) = 365 \cdot Y + Y/4 = 1461 \cdot Y/4$$

Julian calendar

Year Y is a leap year $\iff Y\%4 = 0$.



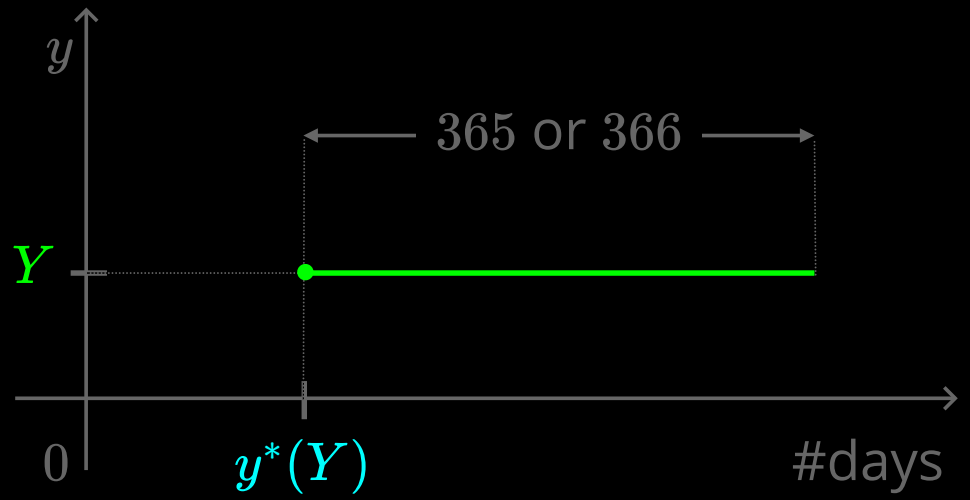
Reasonable attempt:

$$y^*(Y) = 365 \cdot Y + Y/4 = 1461 \cdot Y/4$$

y^* is not a multiplication. ☹

Julian calendar

Year Y is a leap year $\iff Y\%4 = 0$.



Reasonable attempt:

$$y^*(Y) = 365 \cdot Y + Y/4 = 1461 \cdot Y/4$$

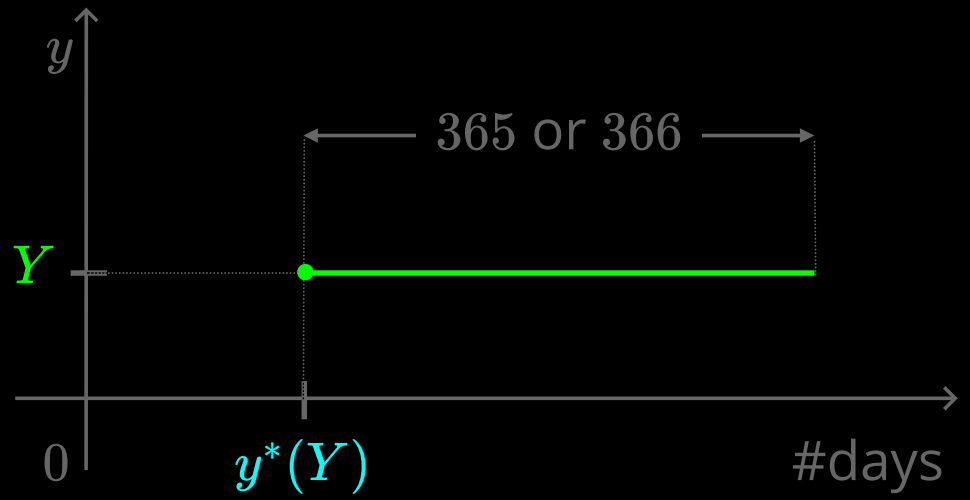
y^* is not a multiplication. ☹

Does not follow the leap year rule. ☹

Y	$y^*(Y)$	
0	0	
1	365	365
2	730	365
3	1095	365
4	1461	366

Julian calendar

Year Y is a leap year $\iff Y\%4 = 0$.



Reasonable attempt:

$$y^*(Y) = 365 \cdot Y + Y/4 = 1461 \cdot Y/4$$

y^* is not a multiplication. ☹

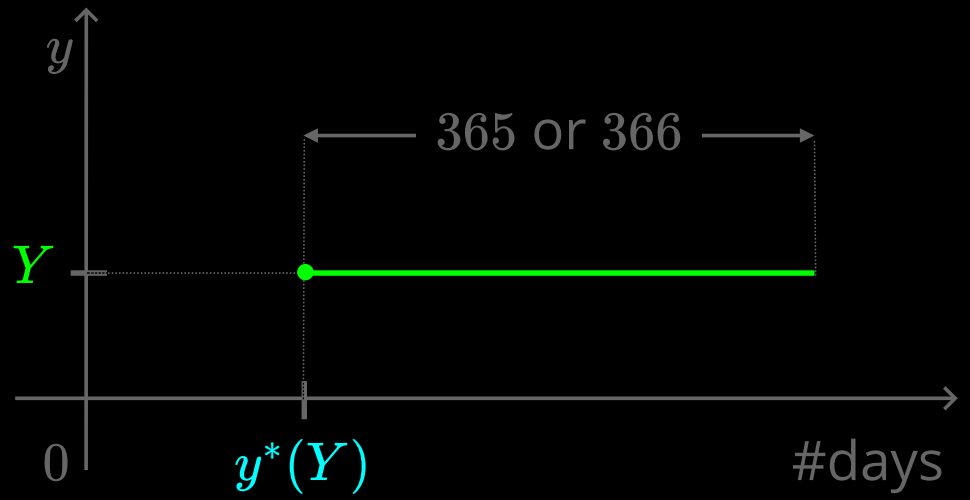
Does not follow the leap year rule. ☹

Y	$y^*(Y)$
0	0
1	365
2	730
3	1095
4	1461

~~365~~
365
365
366

Julian calendar

Year Y is a leap year $\iff Y\%4 = 0$.



Reasonable attempt:

$$y^*(Y) = 365 \cdot Y + Y/4 = 1461 \cdot Y/4$$

y^* is not a multiplication. ☹

Does not follow the leap year rule. ☹

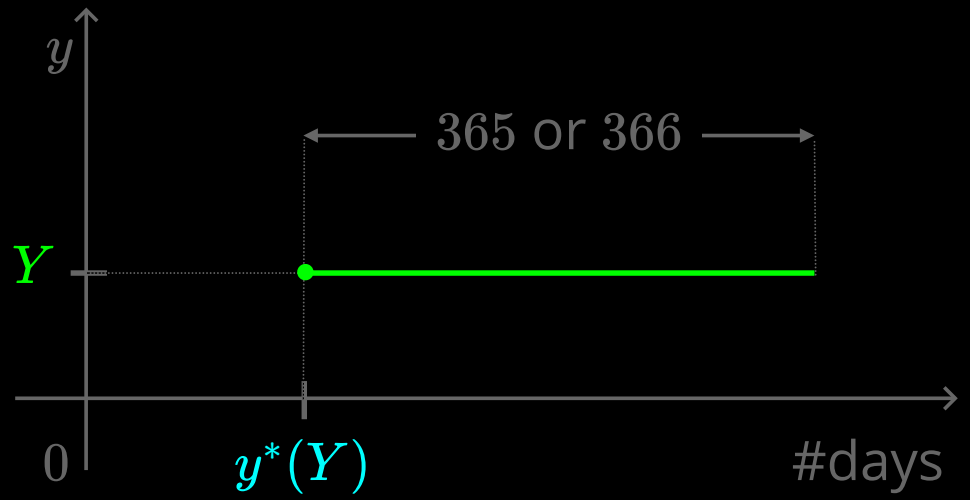
Y	$y^*(Y)$
0	0
1	365
2	730
3	1095
4	1461

~~365~~ 365 365 365 ~~366~~

Ok for $(Y+1)\%4 = 0$.

Julian calendar

Year Y is a leap year $\iff Y\%4 = 0$.



Reasonable attempt:

$$y^*(Y) = 365 \cdot Y + Y/4 = 1461 \cdot Y/4$$

y^* is not a multiplication. ☹

Does not follow the leap year rule. ☹

Y	$y^*(Y)$
0	0
1	365
2	730
3	1095
4	1461

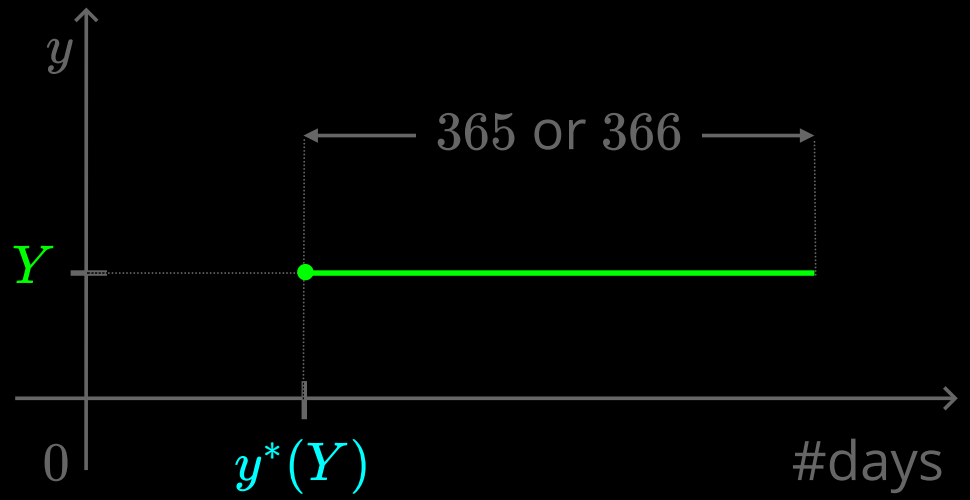
~~365~~ 365 365 ~~366~~

Ok for $(Y+1)\%4 = 0$.

Fix: $y^*(Y) = (1461 \cdot Y + 3)/4$

Julian calendar

Year Y is a leap year $\iff Y\%4 = 0$.



Reasonable attempt:

$$y^*(Y) = 365 \cdot Y + Y/4 = 1461 \cdot Y/4$$

y^* is not a multiplication.

Does not follow the leap year rule.

Y	$y^*(Y)$
0	0
1	365
2	730
3	1095
4	1461

365 ~~365~~ 365 365 ~~366~~

Ok for $(Y+1)\%4 = 0$.

Fix: ~~$y^*(Y) = (1461 \cdot Y + 3)/4$~~

We can do better...

Euclidean affine functions (EAFs)

General case	Particular case ($a = 1$, $b = 0$)
$f(n) = (a \cdot n + b)/d$	$f(n) = n/d$
$f^{\circ}(n) =$???	$f^{\circ}(n) = n\%d$
$f^{*}(q) =$???	$f^{*}(q) = d \cdot q$

Euclidean affine functions (EAFs)

General case	Particular case ($a = 1$, $b = 0$)
$f(n) = (a \cdot n + b)/d$	$f(n) = n/d$
$f^{\circ}(n) = (a \cdot n + b)\%d/a$	$f^{\circ}(n) = n\%d$
$f^{*}(q) = (d \cdot q + a - b - 1)/a$	$f^{*}(q) = d \cdot q$

$$f^{\circ}(n) = n - f^{*}(f(n))$$

$$f^{*}(q) = \min\{\, n \,;\; f(n) = q \,\}$$

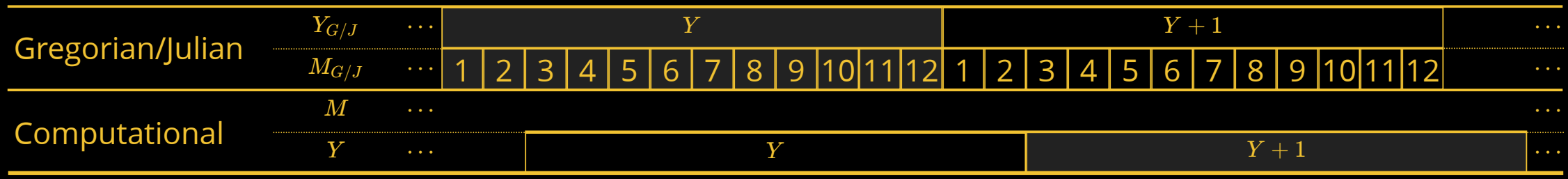
Computational calendar

- Introduced by Christian Zeller in 1882.

Gregorian/Julian	$Y_{G/J}$	$\cdots$	Y											$Y+1$										$\cdots$
	$M_{G/J}$	$\cdots$	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
Computational	M	$\cdots$																									$\cdots$
	Y	$\cdots$																									$\cdots$

Computational calendar

- Introduced by Christian Zeller in 1882.



Computational calendar

- Introduced by Christian Zeller in 1882.

Gregorian/Julian	$Y_{G/J}$	$\cdots$	Y											$Y+1$											$\cdots$
	$M_{G/J}$	$\cdots$	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
Computational	M	$\cdots$			3	4	5	6	7	8	9	10	11	12			3	4	5	6	7	8	9	10	11	12	$\cdots$
	Y	$\cdots$			Y												$Y+1$										$\cdots$

Computational calendar

- Introduced by Christian Zeller in 1882.

Gregorian/Julian	$Y_{G/J}$	$\cdots$	Y												$Y+1$																$\cdots$
	$M_{G/J}$	$\cdots$	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12					$\cdots$
Computational	M	$\cdots$			3	4	5	6	7	8	9	10	11	12	13	14	3	4	5	6	7	8	9	10	11	12	13	14			$\cdots$
	Y	$\cdots$			Y												$Y+1$														$\cdots$

Computational calendar

- Introduced by Christian Zeller in 1882.

Gregorian/Julian	$Y_{G/J}$	$\cdots$	Y												$Y+1$															$\cdots$
	$M_{G/J}$	$\cdots$	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12				$\cdots$
Computational	M	$\cdots$			3	4	5	6	7	8	9	10	11	12	13	14	3	4	5	6	7	8	9	10	11	12	13	14		$\cdots$
	Y	$\cdots$			Y												$Y+1$													$\cdots$

If $M < 13$

$$Y_{G/J} = Y$$
$$M_{G/J} = M$$
$$D_{G/J} = D + 1$$

If $M \geq 13$

$$Y_{G/J} = Y + 1$$
$$M_{G/J} = M - 12$$
$$D_{G/J} = D + 1$$

Computational calendar

- Introduced by Christian Zeller in 1882.

If $M < 13$

$$Y_{G/J} = Y$$
$$M_{G/J} = M$$
$$D_{G/J} = D + 1$$

If $M \geq 13$

$$Y_{G/J} = Y + 1$$
$$M_{G/J} = M - 12$$
$$D_{G/J} = D + 1$$

- Y is a computational leap year $\iff$ $Y + 1$ is a Gregorian/Julian leap year.

Computational calendar (Julian)

Year Y is a leap year $\iff (Y+1)\%4 = 0$.

$$y^*(Y) = 1461 \cdot Y/4$$

Computational calendar (Julian)

Year Y is a leap year $\iff (Y+1)\%4 = 0$.

$$y^*(Y) = 1461 \cdot Y/4$$

$$= (d \cdot Y + a - b - 1)/a \qquad a = 4,\ d = 1461 \text{ and } b = 3$$

Computational calendar (Julian)

Year Y is a leap year $\iff (Y+1)\%4 = 0$.

$$
\begin{aligned}
y^*(Y) &= 1461 \cdot Y/4 \\
&= (d \cdot Y + a - b - 1)/a \qquad\qquad a = 4,\ d = 1461 \text{ and } b = 3 \\
y(N) &= (a \cdot N + b)/d \\
&= (4 \cdot N + 3)/1461 \\
y^\circ(N) &= (a \cdot N + b)\%d/a \\
&= (4 \cdot N + 3)\%1461/4
\end{aligned}
$$

Computational calendar (Gregorian) - I

Year Y is a leap year $\iff (Y+1)\%4 = 0$ but $(Y+1)\%100 \neq 0$, unless $(Y+1)\%400 = 0$.

Number of days before year Y:

$y^*(Y) = 365 \cdot Y + Y/4 - Y/100 + Y/400.$

Computational calendar (Gregorian) - I

Year Y is a leap year $\iff (Y+1)\%4 = 0$ but $(Y+1)\%100 \neq 0$, unless $(Y+1)\%400 = 0$.

Number of days before year Y:

~~$y^*(Y) = 365 \cdot Y + Y/4 - Y/100 + Y/400.$~~

☹ y^* is correct but it's not an EAF!

Computational calendar (Gregorian) - I

Year Y is a leap year $\iff (Y+1)\%4 = 0$ but $(Y+1)\%100 \neq 0$, unless $(Y+1)\%400 = 0$.

Number of days before year Y:

~~$y^*(Y) = 365 \cdot Y + Y/4 - Y/100 + Y/400.$~~

☹ y^* is correct but it's not an EAF!

Fix: introduce century.

Century C is a *leap century* $\iff (C+1)\%4 = 0$.

Computational calendar (Gregorian) - I

Year Y is a leap year $\iff (Y+1)\%4 = 0$ but $(Y+1)\%100 \neq 0$, unless $(Y+1)\%400 = 0$.

Number of days before year Y:

~~$y^*(Y) = 365 \cdot Y + Y/4 - Y/100 + Y/400.$~~

☹ y^* is correct but it's not an EAF!

Fix: introduce century.

Century C is a *leap century* $\iff (C+1)\%4 = 0$.

$c^*(C) =$ #days before century C

$= 36524 \cdot C + C/4$

$= 146097 \cdot C/4$

Computational calendar (Gregorian) - I

Year Y is a leap year $\iff (Y+1)\%4 = 0$ but $(Y+1)\%100 \neq 0$, unless $(Y+1)\%400 = 0$.

Number of days before year Y:

~~$y^*(Y) = 365 \cdot Y + Y/4 - Y/100 + Y/400.$~~

☹ y^* is correct but it's not an EAF!

Fix: introduce century.

Century C is a *leap century* $\iff (C+1)\%4 = 0$.

$c^*(C) =$ #days before century C
$= 36524 \cdot C + C/4$
$= 146097 \cdot C/4$

$c(N) =$ century $= C$
$= (4 \cdot N + 3)/146097$

$c^\circ(N) =$ day of century $= N_C$
$= (4 \cdot N + 3)\%146097/4$

Computational calendar (Gregorian) - II

Date N days after 0000 - 03 - 01:

$$C = (4 \cdot N + 3)/146097$$

$$N_C = (4 \cdot N + 3)\%146097/4$$

Computational calendar (Gregorian) - II

Date N days after 0000 - 03 - 01:

$$C = (4 \cdot N + 3)/146097$$
$$N_C = (4 \cdot N + 3)\%146097/4$$

N_C is within a century: Julian leap rule applies.

$$Z = (4 \cdot N_C + 3)/1461 \qquad \text{(year of the century)}$$
$$N_Y = (4 \cdot N_C + 3)\%1461/4$$

Computational calendar (Gregorian) - II

Date N days after 0000 - 03 - 01:

$$C = (4 \cdot N + 3)/146097$$
$$N_C = (4 \cdot N + 3)\%146097/4$$

N_C is within a century: Julian leap rule applies.

$$Z = (4 \cdot N_C + 3)/1461$$ (year of the century)
$$N_Y = (4 \cdot N_C + 3)\%1461/4$$

$$Y = 100 \cdot C + Z$$

😀 Only month and day remain.

Month and day

M		#days	$m^*(M)$
3	Mar	31	0
4	Apr	30	31
5	May	31	61
6	Jun	30	92
7	Jul	31	122

M		#days	$m^*(M)$
8	Aug	31	153
9	Sep	30	184
10	Oct	31	214
11	Nov	30	245
12	Dec	31	275

M		#days	$m^*(M)$
13	Jan	31	306
14	Fev	28/29	337

Month and day

M		#days	$m^*(M)$
3	Mar	31	0
4	Apr	30	31
5	May	31	61
6	Jun	30	92
7	Jul	31	122

M		#days	$m^*(M)$
8	Aug	31	153
9	Sep	30	184
10	Oct	31	214
11	Nov	30	245
12	Dec	31	275

M		#days	$m^*(M)$
13	Jan	31	306
14	Fev	28/29	337

February at the end $\Longrightarrow$ $m^*(M)$ does not depend on whether the year is leap or not.

Month and day

M		#days	$m^*(M)$
3	Mar	31	0
4	Apr	30	31
5	May	31	61
6	Jun	30	92
7	Jul	31	122

M		#days	$m^*(M)$
8	Aug	31	153
9	Sep	30	184
10	Oct	31	214
11	Nov	30	245
12	Dec	31	275

M		#days	$m^*(M)$
13	Jan	31	306
14	Fev	28/29	337

February at the end $\implies$ $m^*(M)$ does not depend on whether the year is leap or not.

$$m^*(M) = (153 \cdot M - 457)/5$$

$$M = m(N_Y) = (5 \cdot N_Y + 461)/153$$

$$D = m^\circ(N_Y) = (5 \cdot N_Y + 461)\%153/5$$

All together

Date N days after 0000 - 03 - 01:

Computational calendar

$$C = (4 \cdot N + 3)/146097$$
$$N_C = (4 \cdot N + 3)\%146097/4$$

$$Z = (4 \cdot N_C + 3)/1461$$
$$N_Y = (4 \cdot N_C + 3)\%1461/4$$
$$Y = 100 \cdot C + Z$$

$$M = (5 \cdot N_Y + 461)/153$$
$$D = (5 \cdot N_Y + 461)\%153/5$$

Computational ➜ Gregorian

If $M < 13$

$$Y_G = Y$$
$$M_G = M$$
$$D_G = D + 1$$

else

$$Y_G = Y + 1$$
$$M_G = M - 12$$
$$D_G = D + 1$$

Reciprocal algorithm

Number of days from 0000 - 03 - 01 to a given date Y_G - M_G - D_G:

Reciprocal algorithm

Number of days from 0000 - 03 - 01 to a given date Y_G - M_G - D_G:

Gregorian ➜ Computational

Computational calendar

If $M_G > 2$

$$Y = Y_G$$
$$M = M_G$$
$$D = D_G - 1$$

else

$$Y = Y_G - 1$$
$$M = M_G + 12$$
$$D = D_G - 1$$

Reciprocal algorithm

Number of days from 0000 - 03 - 01 to a given date Y_G - M_G - D_G:

Gregorian ➜ Computational

If $M_G > 2$

$$Y = Y_G$$
$$M = M_G$$
$$D = D_G - 1$$

else

$$Y = Y_G - 1$$
$$M = M_G + 12$$
$$D = D_G - 1$$

Computational calendar

Reciprocal algorithm

Number of days from 0000 - 03 - 01 to a given date Y_G - M_G - D_G:

Gregorian ➜ Computational

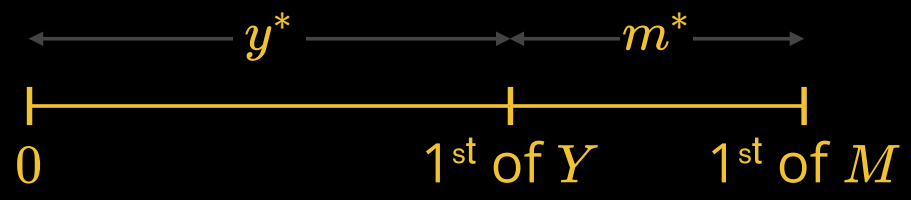
If $M_G > 2$

$$
\begin{aligned}
Y &= Y_G \\
M &= M_G \\
D &= D_G - 1
\end{aligned}
$$

else

$$
\begin{aligned}
Y &= Y_G - 1 \\
M &= M_G + 12 \\
D &= D_G - 1
\end{aligned}
$$

Computational calendar



Reciprocal algorithm

Number of days from 0000 - 03 - 01 to a given date Y_G - M_G - D_G:

Gregorian ➜ Computational

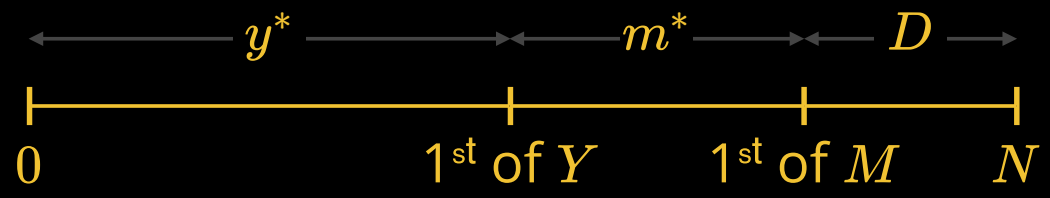
If $M_G > 2$

$$
\begin{aligned}
Y &= Y_G \\
M &= M_G \\
D &= D_G - 1
\end{aligned}
$$

else

$$
\begin{aligned}
Y &= Y_G - 1 \\
M &= M_G + 12 \\
D &= D_G - 1
\end{aligned}
$$

Computational calendar



Reciprocal algorithm

Number of days from 0000 - 03 - 01 to a given date Y_G - M_G - D_G:

Gregorian ➜ Computational

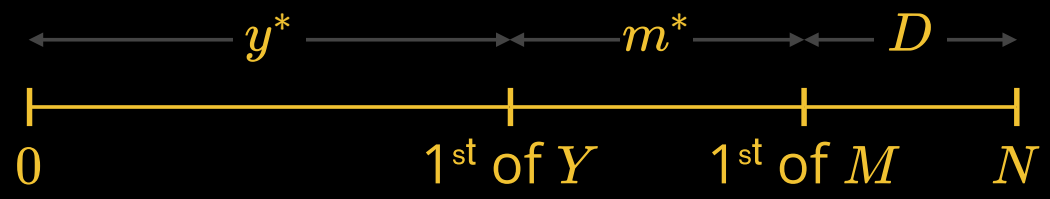
If $M_G > 2$

$$
\begin{aligned}
Y &= Y_G \\
M &= M_G \\
D &= D_G - 1
\end{aligned}
$$

else

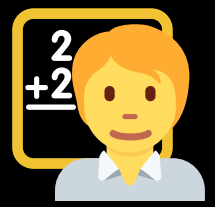
$$
\begin{aligned}
Y &= Y_G - 1 \\
M &= M_G + 12 \\
D &= D_G - 1
\end{aligned}
$$

Computational calendar



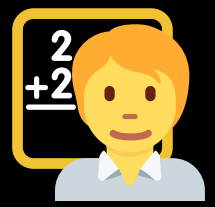
$$
\begin{aligned}
y^* &= 1461 \cdot Y/4 - Y/100 + Y/400 \\
m^* &= (153 \cdot M - 457)/5 \\
N &= y^* + m^* + D
\end{aligned}
$$

Optimising divisions - I

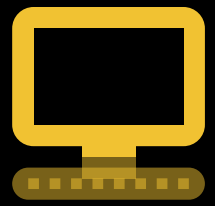


$$\frac{n}{5} = \frac{\frac{10}{5} \cdot n}{10} = \frac{2 \cdot n}{10}$$

Optimising divisions - I

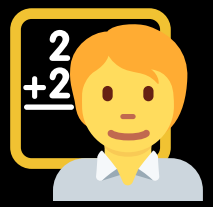


$$\frac{n}{5} = \frac{\frac{10}{5} \cdot n}{10} = \frac{2 \cdot n}{10}$$

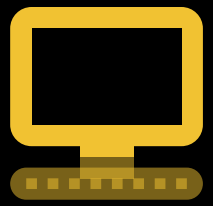


$$\frac{n}{5} = \frac{\frac{2^{34}}{5} \cdot n}{2^{34}} \approx \frac{3435973837 \cdot n}{2^{34}}$$

Optimising divisions - I



$$\frac{n}{5} = \frac{\frac{10}{5} \cdot n}{10} = \frac{2 \cdot n}{10}$$



$$\frac{n}{5} = \frac{\frac{2^{34}}{5} \cdot n}{2^{34}} \approx \frac{3435973837 \cdot n}{2^{34}}$$

Granlund & Montgomery (1994):

😀 $n/5 = 3435973837 \cdot n/2^{34} \quad \forall n \in [0, 2^{32}[$

Optimising divisions - II

```
M  = (5 * N_Y + 461) / 153;
D  = N_Y - (153 * M - 457) / 5;
```

#multiplications

4

Optimising divisions - II

```
M  = (5 * N_Y + 461) / 153;
D  = N_Y - (153 * M - 457) / 5;
```

#multiplications

4

Optimising divisions - II

#multiplications

```
M  = (5 * N_Y + 461) / 153;
D  = N_Y - (153 * M - 457) / 5;
```

4

```
N_3 = 5 * N_Y + 461;
M   = N_3 / 153;
D   = N_3 % 153 / 5;
```

3

Optimising divisions - II

```
M  = (5 * N_Y + 461) / 153;
D  = N_Y - (153 * M - 457) / 5;
```

#multiplications

4

```
N_3 = 5 * N_Y + 461;
M   = N_3 / 153;
//D = N_3 % 153 / 5;
D   = (N_3 - 153 * M) / 5;
```

~~3~~ 4

$$N_3 \% 153 = N_3 - 153 \cdot (N_3/153)$$

Generalisation of GM to EAFs

$$(5 \cdot N_Y + 461)/153 = (2141 \cdot N_Y + 197913)/2^{16}$$

$$(5 \cdot N_Y + 461)\%153/5 = (2141 \cdot N_Y + 197913)\%2^{16}/2141$$

$$\forall N_Y \in [0, 734[$$

Generalisation of GM to EAFs

$$(5 \cdot N_Y + 461)/153 = (2141 \cdot N_Y + 197913)/2^{16}$$

$$(5 \cdot N_Y + 461)\%153/5 = (2141 \cdot N_Y + 197913)\%2^{16}/2141$$

$$\forall N_Y \in [0, 734[$$

```
N_3 = 2141 * N_Y + 197913;
M   = N_3 / 65536;
D   = N_3 % 65536 / 2141;
```

#multiplications

2

Final version

Date N days after 0000 - 03 - 01:

Computational calendar

$$C = (4 \cdot N + 3)/146097$$
$$N_C = (4 \cdot N + 3)\%146097/4$$

$$P_2 = 2939745 \cdot (4 \cdot N_C + 3)$$
$$Z = P_2/2^{32}$$
$$N_Y = P_2\%2^{32}/2939745/4$$
$$Y = 100 \cdot C + Z$$

$$M = (2141 \cdot N_Y + 197913)/2^{16}$$
$$D = (2141 \cdot N_Y + 197913)\%2^{16}/2141$$

Computational ➜ Gregorian

If $M < 13$

$$Y_G = Y$$
$$M_G = M$$
$$D_G = D + 1$$

else

$$Y_G = Y + 1$$
$$M_G = M - 12$$
$$D_G = D + 1$$

Historical bit twiddling

$$N_C = (4 \cdot N + 3)\%146097/4$$

$$P_2 = 2939745 \cdot (4 \cdot N_C + 3)$$

Historical bit twiddling

$$N_C = R/4$$

$$P_2 = 2939745 \cdot (4 \cdot N_C + 3)$$

Historical bit twiddling

$$N_C = R/4$$

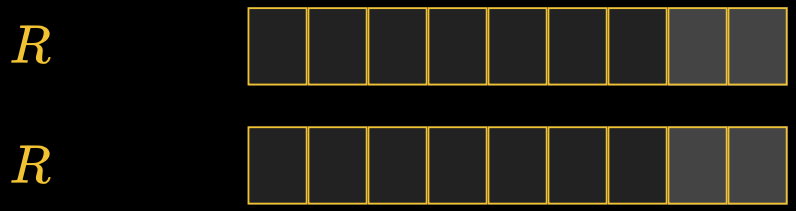
$$4 \cdot N_C + 3$$

Historical bit twiddling

$$4 \cdot (R/4) + 3$$

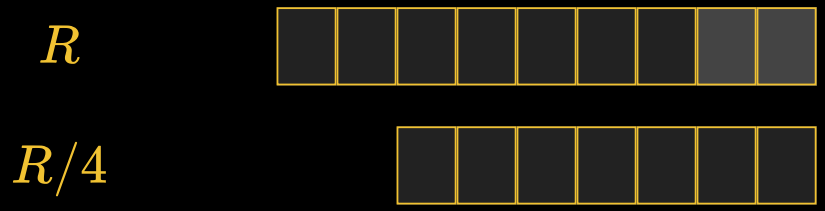
Historical bit twiddling

$$4 \cdot (R/4) + 3$$



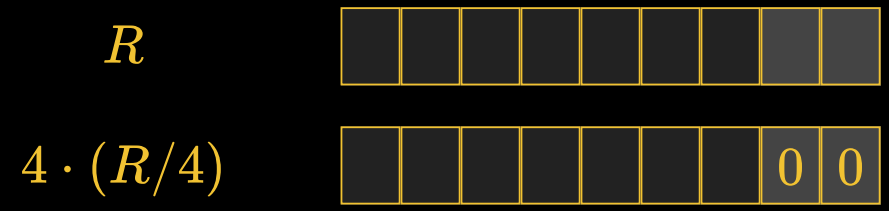
Historical bit twiddling

$$4 \cdot (R/4) + 3$$



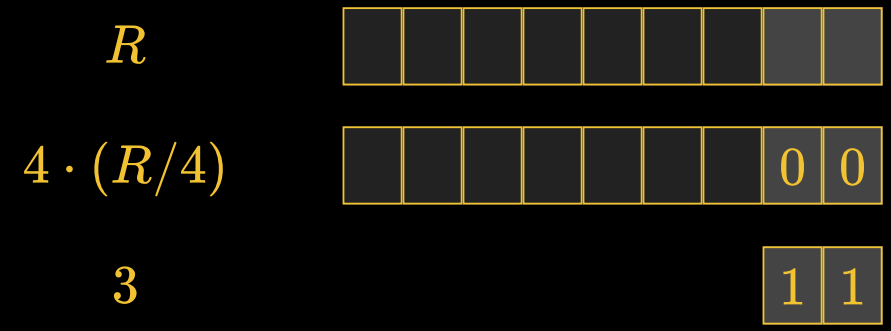
Historical bit twiddling

$$4 \cdot (R/4) + 3$$



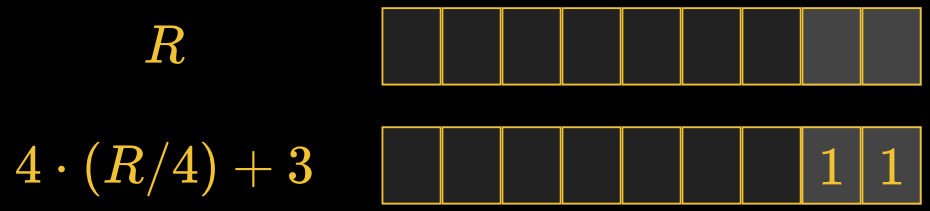
Historical bit twiddling

$$4 \cdot (R/4) + 3$$



Historical bit twiddling

$$4 \cdot (R/4) + 3$$



Historical bit twiddling

$$4 \cdot (R/4) + 3 \;\; = R \mid 3$$

Historical bit twiddling

$$4 \cdot (R/4) + 3 \;\; = R \mid 3$$

Sosigenes 45 BCE

Historical bit twiddling

$$4 \cdot (R/4) + 3 \;\; = R \mid 3$$

Sosigenes 45 BCE

Lilius 1582

Historical bit twiddling

$$4 \cdot (R/4) + 3 \quad = R \mid 3$$

Sosigenes 45 BCE

Lilius 1582

Zeller 1882

Historical bit twiddling

Neri Schneider 2021

$$4 \cdot (R/4) + 3 = R \mid 3$$

Sosigenes 45 BCE

Lilius 1582

Zeller 1882

Thank you

Cristina Acosta
Manuel Caicoya
Daniel Lemire
Becky Rawlings
Lorenz Schneider
Anonymous referee

References

📖 Cassio Neri and Lorenz Schneider
Euclidean affine functions and their application to calendar algorithms
Software Practice and Experience (2022)

📖 https://doi.org/10.1002/spe.3172

💻 https://github.com/cassioneri/eaf