

# Package ‘calcal’

May 8, 2026

**Title** Calendrical Calculations

**Version** 1.0.4

**Description** An R implementation of the algorithms described in Reingold and Dershowitz (4th ed., Cambridge University Press, 2018) <[doi:10.1017/9781107415058](https://doi.org/10.1017/9781107415058)>, allowing conversion between many different calendar systems. Cultural and religious holidays from several calendars can be calculated.

**License** Apache License (>= 2)

**ByteCompile** TRUE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Depends** R (>= 4.1.0)

**Imports** vctrs

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0), tibble

**Config/testthat/edition** 3

**URL** <https://pkg.robjhyndman.com/calcal/>,  
<https://github.com/robjhyndman/calcal>

**VignetteBuilder** knitr

**BugReports** <https://github.com/robjhyndman/calcal/issues>

**NeedsCompilation** no

**Author** Rob Hyndman [aut, cre, cph] (ORCID:  
<<https://orcid.org/0000-0002-2140-5352>>),  
Edward Reingold [cph, ctb] (Original Lisp code),  
Nachum Dershowitz [cph, ctb] (Original Lisp code)

**Maintainer** Rob Hyndman <Rob.Hyndman@monash.edu>

**Repository** CRAN

**Date/Publication** 2026-02-28 23:30:08 UTC

## Contents

advent . . . . .	3
as_date . . . . .	4
as_time_of_day . . . . .	4
babylonian_date . . . . .	5
bahai_date . . . . .	6
bahai_new_year . . . . .	7
balinese_date . . . . .	8
chinese_date . . . . .	9
chinese_new_year . . . . .	11
coptic_date . . . . .	11
day_of_week . . . . .	12
egyptian_date . . . . .	14
french_date . . . . .	15
granularity_names . . . . .	16
gregorian_date . . . . .	17
hebrew_date . . . . .	18
hindu_lunar_new_year . . . . .	19
hindu_solar_date . . . . .	20
icelandic_date . . . . .	21
islamic_date . . . . .	22
islamic_new_year . . . . .	23
iso_date . . . . .	24
julian_date . . . . .	25
kajeng_keliwon . . . . .	26
location . . . . .	27
lunar_phase . . . . .	27
mayan_date . . . . .	28
new_calendar . . . . .	29
new_moons . . . . .	31
persian_date . . . . .	32
roman_date . . . . .	33
sunrise . . . . .	34
tibetan_date . . . . .	34
tibetan_new_year . . . . .	35
time_of_day . . . . .	36
us_memorial_day . . . . .	37
yom_kippur . . . . .	38

## Index

40

---

advent	<i>Christian Ecclesiastical Holidays</i>
--------	--

---

**Description**

Functions to return Gregorian dates for various Christian ecclesiastical holidays and other special days

Coptic Christmas is celebrated on 29th of Koiak in the Coptic calendar, which currently corresponds to 7 or 8 January in the Gregorian calendar.

**Usage**

`advent(year)`

`christmas(year)`

`orthodox_christmas(year)`

`epiphany(year)`

`easter(year)`

`orthodox_easter(year)`

`pentecost(year)`

`coptic_christmas(year)`

`astronomical_easter(year)`

**Arguments**

`year`                   Gregorian year

**Value**

A vector of dates on the Gregorian calendar

**Examples**

```
tibble::tibble(
  year = 2025:2030,
  advent = advent(year),
  christmas = christmas(year),
  orthodox_christmas = orthodox_christmas(year),
  epiphany = epiphany(year),
  easter = easter(year),
  orthodox_easter = orthodox_easter(year),
```

```

    pentecost = pentecost(year)
  )

```

---

as\_date

*Create a new date vector or convert a date vector to a new calendar*


---

### Description

New dates can be calculated using `new_date()` for any calendar. Dates can be converted from one calendar to another using `as_date()`. `as_date()` also works with the native R `Date` class and several other classes. When applied to integers, the conversion is from the RD day number (with day 1 being 01-01-01 on the Gregorian calendar).

### Usage

```
as_date(date, calendar)
```

```
new_date(..., calendar)
```

### Arguments

date	Date vector on some calendar
calendar	Target calendar of class "calendar"
...	Named arguments denoting the granularities required for calendar.

### Value

A date vector of class "rdvec" with the specified calendar.

### Examples

```

april2025 <- new_date(year = 2025, month = 4, day = 1:30, calendar = cal_gregorian)
as_date(april2025, calendar = cal_iso)

```

---

as\_time\_of\_day

*Convert to time of day*


---

### Description

Convert to time of day

### Usage

```
as_time_of_day(x, ...)
```

**Arguments**

x                    Vector of times  
 ...                  Additional arguments not currently used

**Value**

A vector containing "time\_of\_day" objects

**See Also**

[time\\_of\\_day](#)

**Examples**

```
as_time_of_day(Sys.time())
```

---

babylonian_date	<i>Babylonian calendar dates</i>
-----------------	----------------------------------

---

**Description**

The classical Babylonian calendar was a lunisolar calendar with a fixed 19-year Metonic cycle.

**Usage**

```
babylonian_date(
  year = integer(),
  month = integer(),
  leap_month = logical(),
  day = integer()
)

as_babylonian(date)
```

**Arguments**

year                Numeric vector of years  
 month              Numeric vector of months  
 leap\_month        Logical vector of leap months  
 day                Numeric vector of days  
 date                Vector of dates on some calendar.

**Value**

A babylonian vector object

**See Also**[cal\\_babylonian](#)**Examples**

```
## Not run:
tibble::tibble(
  gregorian = gregorian_date(2335, 1, 1:2),
  babylonian = as_babylonian(gregorian)
)
babylonian_date(2335, 6, FALSE, 1:2)

## End(Not run)
```

---

`bahai_date`*Bahá'í calendar dates*

---

**Description**

The Bahá'í calendar is a solar calendar used in the Bahá'í faith comprising 18 months, with four or five intercalary days each year. The New Year is at the northern Spring equinox, corresponding to 21 March on the Gregorian calendar. Ayyám-i-Há is specified as month 20.

**Usage**

```
bahai_date(
  major = integer(),
  cycle = integer(),
  year = integer(),
  month = integer(),
  day = integer()
)

as_bahai(date)
```

**Arguments**

<code>major</code>	A numeric vector of major periods
<code>cycle</code>	A numeric vector of cycles
<code>year</code>	A numeric vector of years within the cycles
<code>month</code>	A numeric vector of months
<code>day</code>	A numeric vector of days
<code>date</code>	A numeric vector of dates

**Value**

A bahai vector object

**See Also**

[cal\\_bahai](#), [bahai\\_new\\_year](#)

**Examples**

```
tibble::tibble(  
  gregorian = gregorian_date(2025, 2, 15) + 0:30,  
  bahai = as_bahai(gregorian)  
)  
bahai_date(1, 10, 11, 3, 5:7)
```

---

bahai_new_year	<i>Bahá'í holidays</i>
----------------	------------------------

---

**Description**

Dates are returned as Gregorian dates

**Usage**

```
bahai_new_year(year)  
  
naw_ruz(year)  
  
feast_of_ridvan(year)  
  
birth_of_the_bab(year)
```

**Arguments**

year                    The year on the Gregorian calendar

**Value**

A vector of dates on the Gregorian calendar

**See Also**

[bahai\\_date](#)

**Examples**

```
tibble::tibble(  
  year = 2025:2030,  
  new_year = bahai_new_year(year),  
  naw_ruz = naw_ruz(year),  
  ridvan = feast_of_ridvan(year),  
  birth_bab = birth_of_the_bab(year)  
)
```

---

balinese\_date                      *Balinese Pawukon calendar dates*

---

### Description

The Balinese calendar repeats every 210 days. It has 10 concurrent weeks, of lengths 1, 2, ..., 10 days. The 210 day cycles are unnumbered, so there is no way to convert a Balinese date into a unique date on another calendar.

### Usage

```
balinese_date(  
  luang = integer(),  
  dwiwara = integer(),  
  triwara = integer(),  
  caturwara = integer(),  
  pancawara = integer(),  
  sadwara = integer(),  
  saptawara = integer(),  
  asatawara = integer(),  
  sangawara = integer(),  
  dasawara = integer()  
)
```

```
as_balinese(date)
```

### Arguments

luang	A numeric vector
dwiwara	A numeric vector
triwara	A numeric vector
caturwara	A numeric vector
pancawara	A numeric vector
sadwara	A numeric vector
saptawara	A numeric vector
asatawara	A numeric vector
sangawara	A numeric vector
dasawara	A numeric vector
date	A vector of dates on some calendar.

### Value

A balinese vector object

**See Also**[kajeng\\_keliwon](#)**Examples**

```
gregorian_date(2025, 6, 1:10) |>
  as_balinese()
```

---

`chinese_date`*Chinese, Japanese, Korean and Vietnamese calendar dates*

---

**Description**

The traditional Chinese lunisolar calendar uses a 60-year cycle with 12 months per year. The Japanese, Korean and Vietnamese calendars are almost identical, but with different locations for determining astronomical positions.

**Usage**

```
chinese_date(
  cycle = integer(),
  year = integer(),
  month = integer(),
  leap_month = logical(),
  day = integer()
)
```

```
japanese_date(
  cycle = integer(),
  year = integer(),
  month = integer(),
  leap_month = logical(),
  day = integer()
)
```

```
korean_date(
  year = integer(),
  month = integer(),
  leap_month = logical(),
  day = integer()
)
```

```
vietnamese_date(
  cycle = integer(),
  year = integer(),
  month = integer(),
  leap_month = logical(),
)
```

```

    day = integer()
  )

as_chinese(date)

as_japanese(date)

as_korean(date)

as_vietnamese(date)

```

### Arguments

cycle	A numeric vector of cycles
year	A numeric vector of years within the cycles
month	A numeric vector of months
leap_month	A logical vector indicating leap months
day	A numeric vector of days
date	A numeric vector of dates

### Value

A chinese vector object

### See Also

[cal\\_chinese](#), [chinese\\_new\\_year](#)

### Examples

```

chinese <- new_date(
  cycle = 78, year = 42, month = 5, leap_month = FALSE, day = 16:18,
  calendar = cal_chinese
)
chinese
chinese_date(78, 42, 5, FALSE, 16:18)
as_date(chinese, calendar = cal_gregorian)
as_date(Sys.Date(), calendar = cal_chinese)
tibble::tibble(
  gregorian = gregorian_date(2025, 1, 1) + 0:364,
  chinese = as_chinese(gregorian)
)
as_gregorian(chinese_date(78, 41, 12, FALSE, 3:30))
as_chinese(gregorian_date(2025, 1, 1:28))
as_chinese("2016-01-01")
as_chinese(Sys.Date())

```

---

chinese_new_year	<i>Chinese holidays</i>
------------------	-------------------------

---

**Description**

Dates are returned as Gregorian dates

**Usage**

```
chinese_new_year(year)
```

```
dragon_festival(year)
```

```
qing_ming(year)
```

**Arguments**

year                    The year on the Gregorian calendar

**Value**

A vector of dates on the Gregorian calendar

**See Also**

[chinese\\_date](#)

**Examples**

```
tibble::tibble(  
  year = 2025:2030,  
  cny = chinese_new_year(year),  
  qm = qing_ming(year),  
  dbf = dragon_festival(year)  
)
```

---

coptic_date	<i>Coptic and Ethiopic calendar dates</i>
-------------	---

---

**Description**

These two calendars are identical apart from the starting point or epoch. The Coptic calendar (also called the Alexandrian calendar) starts on 29 August 284 CE in the Julian calendar, while the Ethiopic (or Ethiopian) calendar starts on 29 August 8 CE in the Julian calendar. The Coptic calendar is used by the Coptic Orthodox and Coptic Catholic Churches, while the Ethiopic calendar is the official state calendar of Ethiopia, and unofficial calendar of Eritrea, and is used by the Ethiopian and Eritrean Orthodox Churches. Both calendars have 13 months, with 12 months of 30 days and a 13th month of 5 or 6 days depending on whether it is a leap year. Leap years occur every 4 years.

**Usage**

```
coptic_date(year = integer(), month = integer(), day = integer())
```

```
ethiopic_date(year = integer(), month = integer(), day = integer())
```

```
as_coptic(date)
```

```
as_ethiopic(date)
```

**Arguments**

year	A numeric vector of years
month	A numeric vector of months
day	A numeric vector of days
date	A numeric vector of dates

**Value**

A coptic or ethiopic vector object

**See Also**

[cal\\_coptic](#), [cal\\_ethiopic](#), [coptic\\_christmas](#)

**Examples**

```
tibble::tibble(  
  gregorian = gregorian_date(2025, 1, 1:31),  
  coptic = as_coptic(gregorian),  
  ethiopic = as_ethiopic(gregorian)  
)  
coptic_date(1741, 5, 16:18)  
as_date(Sys.Date(), calendar = cal_ethiopic)  
as_coptic("2016-01-01")  
as_ethiopic(Sys.Date())
```

---

day\_of\_week

*Compute granularities from dates*

---

**Description**

Compute days, weeks, or months from a vector of dates. These work for Gregorian dates, and for some other calendars where it makes sense. In particular, `day_of_week` has been implemented for many calendars that contain the concept of a week. Similarly, `day_of_month`, `day_of_year` and `days_remaining` will work for several calendars.

**Usage**

```
day_of_week(date, ...)  
  
day_of_month(date)  
  
day_of_year(date)  
  
days_remaining(date)  
  
week_of_month(date, first_day = "Monday")  
  
week_of_year(date, first_day = "Monday")  
  
month_of_year(date)  
  
year(date)
```

**Arguments**

date	A vector of dates
...	Other arguments used for specific calendars
first_day	Character denoting first day of the week. Default is "Monday"

**Details**

week\_of\_year() returns the ISO 8601 week number with first\_day as Monday. Under this standard, week 1 of a year is defined as the first week with at least 4 days in the year; equivalently, it is the week containing 4 January. There is no week 0; instead week 1 of a year may begin in the previous calendar year.

week\_of\_month() is defined analogously where week 1 of a month is the first week with at least 4 days in the month; equivalently, it is the week containing the 4th day of the month. There is no week 0; instead week 1 of a month may begin in the previous calendar month.

days\_remaining() returns the number of days remaining in the year.

Other functions should be self-explanatory.

**Value**

A vector of numerical values for the requested granularity. In the case of day\_of\_week(), it returns a character vector of the name of the day of the week, or a numeric vector if numeric = TRUE is specified.

**Examples**

```
april2025 <- gregorian_date(2025, 4, 1:30)  
day_of_week(april2025)  
day_of_month(april2025)  
day_of_year(april2025)  
days_remaining(april2025)
```

```
week_of_month(april2025)
week_of_year(april2025)
month_of_year(april2025)
```

---

egyptian\_date

*Egyptian and Armenian calendar dates*

---

## Description

The ancient Egyptian calendar is a 365-day solar calendar with 12 months of 30 days each, plus a 13th month of 5 days. The Armenian calendar is similar but has a different epoch and month names.

## Usage

```
egyptian_date(year = integer(), month = integer(), day = integer())
```

```
armenian_date(year = integer(), month = integer(), day = integer())
```

```
as_egyptian(date)
```

```
as_armenian(date)
```

## Arguments

year	Numeric vector of years
month	Numeric vector of months
day	Numeric vector of days
date	Vector of dates on some calendar

## Value

An egyptian or armenian vector object

## Examples

```
tibble::tibble(
  gregorian = gregorian_date(2025, 5, 1:10),
  egyptian = as_egyptian(gregorian),
  armenian = as_armenian(gregorian)
)
```

---

french_date	<i>French Revolutionary calendar dates</i>
-------------	--

---

## Description

There are two versions of the French Revolutionary Calendar. The original version, used from 1793, was kept in sync with the solar year by setting the first day of Vendemiaire to the autumnal equinox. The second version, proposed in 1795, was a simpler arithmetic calendar, but was never used. We distinguish the two by using "afrench" (for Arithmetic French) for the second form.

## Usage

```
french_date(year = integer(), month = integer(), day = integer())  
afrench_date(year = integer(), month = integer(), day = integer())  
as_french(date)  
as_afrench(date)
```

## Arguments

year	year
month	month
day	day
date	A vector of dates on some calendar

## Value

A vector of dates on the French Revolutionary calendar

## Examples

```
french_date(1, 1, 1:15) |>  
  as_gregorian()  
french_date(1, 1, 1:15) |>  
  day_of_week()
```

---

granularity_names	<i>Canonical granularities</i>
-------------------	--------------------------------

---

### Description

granularities() will return a character vector of canonical granularity names for the relevant calendar. These are the granularities used to define dates on the calendar. granularity() will return a vector of numerical values for a given canonical granularity.

### Usage

```
granularity_names(calendar)
granularity(date, granularity)
```

### Arguments

calendar	An object defining a calendar, of "calendar" class.
date	A date vector on some calendar
granularity	A character string indicating the granularity to extract

### Value

A character vector of granularity names or a vector of numerical values for the specified granularity.

### See Also

[week\\_of\\_year](#) for some non-canonical granularities.

### Examples

```
granularity_names(cal_iso)
granularity_names(cal_gregorian)
date_iso <- new_date(year = 2025, week = 23, day = 2, calendar = cal_iso)
granularity(date_iso, "week")
date_gregorian <- new_date(year = 2025, month = 1, day = 1, calendar = cal_gregorian)
granularity(date_gregorian, "month")
```

---

gregorian_date	<i>Gregorian calendar dates</i>
----------------	---------------------------------

---

## Description

The Gregorian calendar is the standard calendar used by most of the world. It was named for Pope Gregory XIII who introduced it in 1582. It is a modification of the Julian calendar which was in use at the time.

## Usage

```
gregorian_date(year = integer(), month = integer(), day = integer())
```

```
as_gregorian(date)
```

## Arguments

year	A numeric vector of years
month	A numeric vector of months
day	A numeric vector of days
date	Vector of dates on some calendar

## Value

A gregorian vector object

## See Also

[cal\\_gregorian](#)

## Examples

```
new_date(year = 2025, month = 3, day = 2:4, calendar = cal_gregorian)
gregorian_date(2025, 4, 19:30)
as_date(Sys.Date(), calendar = cal_gregorian)
as_gregorian(Sys.Date())
as_gregorian("2016-01-01")
tibble::tibble(
  x = seq(as.Date("2025-01-01"), as.Date("2025-12-31"), by = "day"),
  y = as_gregorian(x),
  z = as_date(x, calendar = cal_gregorian)
)
```

---

hebrew_date	<i>Hebrew calendar dates</i>
-------------	------------------------------

---

### Description

The Hebrew (or Jewish) calendar is an official calendar of Israel, and is used for Jewish religious holidays. It is a lunisolar calendar comprising months of 29 or 30 days, which begin and end at approximately the time of the new moon. An extra lunar month is added every 2 or 3 years, so the calendar has either 12 or 13 months per year.

### Usage

```
hebrew_date(year = integer(), month = integer(), day = integer())
```

```
as_hebrew(date)
```

```
ohebrew_date(year = integer(), month = integer(), day = integer())
```

```
as_ohebrew(date)
```

```
samaritan_date(year = integer(), month = integer(), day = integer())
```

```
as_samaritan(date)
```

### Arguments

year	A numeric vector of years
month	A numeric vector of months
day	A numeric vector of days
date	Vector of dates on some calendar

### Details

The observational Hebrew calendar ("ohebrew") is the classical calendar where the new month began with the reported observation of the crescent new moon. In this implementation, Haifa is taken as the point of observation.

The Samaritan calendar is similar, but the moment of new moon marking the start of each new month is based on a traditional reckoning of the lunar cycle,

### Value

A hebrew vector object

### See Also

[cal\\_hebrew](#), [rosh\\_hashanah](#)

### Examples

```
heb <- new_date(year = 5785, month = 3, day = 2:4, calendar = cal_hebrew)
heb
hebrew_date(5785, 3, 2:4)
as_date(heb, calendar = cal_gregorian)
as_date(Sys.Date(), calendar = cal_hebrew)
tibble::tibble(
  gregorian = gregorian_date(2025, 1, 1) + 0:364,
  hebrew = as_date(gregorian, calendar = cal_hebrew),
)
as_gregorian(hebrew_date(5785, 3, 2:4))
as_hebrew(gregorian_date(2025, 1, 1:31))
as_hebrew("2016-01-01")
as_hebrew(Sys.Date())
hebrew_date(5785, 3, 1:10) |> day_of_week()
```

---

hindu\_lunar\_new\_year *Hindu holidays and special days*

---

### Description

Functions to return Gregorian dates for various Hindu holidays based on the Hindu calendars. Hindu Lunar New Year is the first day of the lunar month of Caitra (month 1). The Birthday of Rama is on the 8th or 9th day of the first month (Caitra). Diwali is on the new moon day in the month of Kartik (month 8). The Great Night of Shiva is at the end of the 11 month of Magha. Mesha Sankranti is the day when the sun enters the sign of Aries (Mesha). Sacred Wednesdays are the 8th day of the lunar month that falls on a Wednesday. Dates may vary by a day or two due to variations of the lunar calendar and local traditions.

### Usage

```
hindu_lunar_new_year(year)
```

```
mesha_sankranti(year)
```

```
diwali(year)
```

```
shiva(year)
```

```
rama(year)
```

```
sacred_wednesdays(year)
```

### Arguments

year                    A numeric vector of Gregorian years

**Value**

A vector of dates on the Gregorian calendar

**See Also**

[hindu\\_lunar\\_date](#)

**Examples**

```
shiva(2025:2026)
hindu_lunar_new_year(2025:2026)
rama(2025:2026)
mesha_sankranti(2025:2026)
diwali(2025:2026)
sacred_wednesdays(2025:2026)
```

---

hindu_solar_date	<i>Hindu solar and lunar calendar dates</i>
------------------	---

---

**Description**

There are four Hindu calendars implemented: modern Hindu solar and lunar calendars, and the old Hindu solar and lunar calendars. Hindu solar months are 1/12 of a solar year (approximately 30.44 days), while lunar months are based on the lunar cycle (approximately 29.53 days).

**Usage**

```
hindu_solar_date(year, month, day)
```

```
hindu_lunar_date(year, month, leap_month, day, leap_day)
```

```
as_hindu_solar(date)
```

```
as_hindu_lunar(date)
```

```
old_hindu_solar_date(year = integer(), month = integer(), day = integer())
```

```
old_hindu_lunar_date(
  year = integer(),
  month = integer(),
  leap_month = logical(),
  day = integer()
)
```

```
as_old_hindu_solar(date)
```

```
as_old_hindu_lunar(date)
```

**Arguments**

year	A numeric vector of years
month	A numeric vector of months
day	A numeric vector of days
leap_month	A logical vector indicating if year is a leap year
leap_day	A logical vector indicating if day is a leap day
date	A date vector on some calendar

**Value**

A vector object of Hindu dates.

**See Also**

[cal\\_hindu\\_solar](#), [cal\\_hindu\\_lunar](#), [cal\\_old\\_hindu\\_solar](#), [cal\\_old\\_hindu\\_lunar](#), [diwali](#)

**Examples**

```
gregorian_date(2025, 1, 1:31) |>  
  as_hindu_solar()  
gregorian_date(2025, 1, 1:31) |>  
  as_hindu_lunar()
```

---

icelandic_date	<i>Icelandic calendar dates</i>
----------------	---------------------------------

---

**Description**

The Icelandic calendar, still in use in Iceland, divides times into 7-day weeks and two seasons: Summer and Winter. Summer starts on the first Thursday after April 18th, and Winter 180 days earlier. Ordinary years have 52 weeks with leap years having 53 weeks. The leap week occurs every 5-7 years in midsummer.

**Usage**

```
icelandic_date(  
  year = integer(),  
  season = integer(),  
  week = integer(),  
  weekday = integer()  
)  
  
as_icelandic(date)
```

**Arguments**

year	A numeric vector of years
season	A numeric vector of seasons (1 = Summer, 2 = Winter)
week	A numeric vector of weeks within the season (1 to 28)
weekday	A number vector containing day of week (0 = Sunday, 1 = Monday, ..., 6 = Saturday)
date	A numeric vector of dates

**Value**

An icelandic vector object

**Examples**

```
gregorian_date(2025, 4, 20:30) |>
  as_icelandic()
icelandic_date(2025, 1, 6, 0:6) |>
  day_of_week()
```

---

islamic_date	<i>Islamic calendar dates</i>
--------------	-------------------------------

---

**Description**

The Islamic (or Hijri) calendar is a lunar calendar comprising 12 lunar months in a year of 354 or 355 days. It is widely used in for Islamic holidays, and in countries where the predominant religion is Islam.

**Usage**

```
islamic_date(year = integer(), month = integer(), day = integer())

as_islamic(date)

oislamic_date(year = integer(), month = integer(), day = integer())

as_oislamic(date)

saudi_date(year = integer(), month = integer(), day = integer())

as_saudi(date)
```

**Arguments**

year	A numeric vector of years
month	A numeric vector of months
day	A numeric vector of days
date	Vector of dates on some calendar

## Details

Three variations are implemented here. The standard Islamic calendar is available using `as_islamic` and `islamic_date`. The Saudi Islamic calendar uses `as_saudi` and `saudi_date`, while the traditional observational Islamic calendar is available using `as_oislamic` and `oislamic_date`.

## Value

An islamic vector object

## See Also

[cal\\_islamic](#), [ramadan](#)

## Examples

```
islamic_date(2025, 5, 1:30)
as_islamic("2016-01-01")
as_islamic(Sys.Date())
tibble::tibble(
  x = seq(as.Date("2025-01-01"), as.Date("2025-12-31"), by = "day"),
  y = as_islamic(x)
)
islamic_date(2025, 5, 1:10) |> day_of_week()
islamic_date(2025, 4, 19:30)
```

---

<code>islamic_new_year</code>	<i>Islamic holidays</i>
-------------------------------	-------------------------

---

## Description

Functions to return Gregorian dates for various Islamic holidays. Specific dates can vary slightly based on moon sightings in different regions.

## Usage

```
islamic_new_year(year)
```

```
mawlid(year)
```

```
ramadan(year)
```

```
eid_al_fitr(year)
```

```
eid_al_adha(year)
```

## Arguments

`year` A numeric vector of Gregorian years

**Value**

A vector of dates on the Gregorian calendar

**See Also**

[islamic\\_date](#)

**Examples**

```
tibble::tibble(
  year = 2025:2029,
  `New year` = islamic_new_year(year),
  Mawlid = mawlid(year),
  Ramadan = ramadan(year),
  `Eid al-Fitr` = eid_al_fitr(year),
  `Eid al-Adha` = eid_al_adha(year)
)
ramadan(2030)
```

---

iso\_date

*ISO calendar dates*

---

**Description**

In ISO 8601 date objects, weeks are defined as starting on Mondays. Week 1 is the first week with at least 4 days in the year. Equivalently, it is the week containing 4 January. There is no week 0; instead week 1 of a year may begin in the previous calendar year.

**Usage**

```
iso_date(year = integer(), week = integer(), day = integer())
```

```
as_iso(date)
```

**Arguments**

year	A numeric vector of years
week	A numeric vector of weeks
day	A numeric vector of days
date	Vector of dates on some calendar

**Details**

More flexible week numbering is possible using Gregorian dates with [week\\_of\\_year\(\)](#).

**Value**

An iso vector object

**See Also**[cal\\_iso](#), [week\\_of\\_year](#)**Examples**

```
iso <- new_date(year = 2025, week = 23, day = 2:4, calendar = cal_iso)
iso
iso_date(2025, 23, 2:4)
as_gregorian(iso_date(2025, 23, 2:4))
as_iso(gregorian_date(2025, 1, 1:31))
as_iso("2016-01-01")
as_iso(Sys.Date())
tibble::tibble(
  x = seq(as.Date("2025-01-01"), as.Date("2025-12-31"), by = "day"),
  y = as_iso(x)
)
```

---

julian_date	<i>Julian calendar dates</i>
-------------	------------------------------

---

**Description**

The Julian calendar is the calendar used by the Roman Empire, and takes its name from Julius Caesar, who introduced it in 46 BC. It is still used as a religious calendar in parts of the Eastern Orthodox Church.

**Usage**

```
julian_date(year = integer(), month = integer(), day = integer())

as_julian(date)
```

**Arguments**

year	A numeric vector of years
month	A numeric vector of months
day	A numeric vector of days
date	Vector of dates on some calendar

**Value**

A julian vector object

**See Also**[cal\\_julian](#)

**Examples**

```
as_date("2016-01-01", calendar = cal_julian)
as_date(Sys.Date(), calendar = cal_julian)
tibble::tibble(
  x = seq(as.Date("2025-01-01"), as.Date("2025-12-31"), by = "day"),
  y = as_date(x, calendar = cal_gregorian),
  z = as_date(x, calendar = cal_julian)
)
new_date(year = 2025, month = 4, day = 19:30, calendar = cal_julian)
julian_date(2025, 4, 19:30)
as_julian("2016-01-01")
as_julian(Sys.Date())
tibble::tibble(
  x = seq(as.Date("2025-01-01"), as.Date("2025-12-31"), by = "day"),
  y = as_julian(x)
)
```

---

kajeng\_keliwon

*Balinese special days*

---

**Description**

Find all occurrences of Kajeng Keliwon and Tumpek in a vector of Gregorian years.

**Usage**

```
kajeng_keliwon(year)
```

```
tumpek(year)
```

**Arguments**

year            A numeric vector of Gregorian years

**Value**

A vector of dates on the Gregorian calendar

**See Also**

[balinese\\_date](#)

**Examples**

```
kajeng_keliwon(2025)
tumpek(2025)
```

---

location	<i>Locations</i>
----------	------------------

---

**Description**

Create a location object. These are used for calculating the timing of astronomical events such as sunrise and sunset.

**Usage**

```
location(  
  latitude = numeric(),  
  longitude = numeric(),  
  elevation = numeric(),  
  zone = numeric()  
)
```

**Arguments**

latitude	A numeric vector of latitudes
longitude	A numeric vector of longitudes
elevation	A numeric vector of elevations above sea level (in metres)
zone	A numeric vector of time zones (in hours, relative to UTC)

**Value**

A location vector object

**Examples**

```
melbourne <- location(-37.8136, 144.9631, 31, 10)  
sunrise("2025-01-01", melbourne)
```

---

lunar_phase	<i>Lunar phase at date</i>
-------------	----------------------------

---

**Description**

Lunar phase at date, as an angle in degrees. An angle of 0 means a new moon, 90 degrees means the first quarter, 180 means a full moon, and 270 degrees means the last quarter.

**Usage**

```
lunar_phase(date)
```

**Arguments**

date            Date vector

**Value**

A numeric vector of angles in degrees representing the lunar phase at the given dates.

**Examples**

```
april2025 <- gregorian_date(2025, 4, 1:30)
lunar_phase(april2025)
```

---

mayan_date	<i>Mayan calendar dates</i>
------------	-----------------------------

---

**Description**

There are three Mayan calendars: the famous "long count" calendar, the "Haab" calendar, and the "Tzolkin" calendar. Of these, only the long count calendar can be converted to and from other calendars, so it is the only one that has been implemented here.

**Usage**

```
mayan_date(
  baktun = integer(),
  katun = integer(),
  tun = integer(),
  uinal = integer(),
  kin = integer()
)
```

```
as_mayan(date)
```

**Arguments**

baktun            Numeric vector  
katun             Numeric vector  
tun                Numeric vector  
uinal             Numeric vector  
kin                Numeric vector  
date               Vector of dates on some calendar

**Details**

The Mayan long count calendar is a vigesimal (base-20) calendar with five components: kin (1 day), uinal (20 kin), tun (18 uinal), katun (20 tun), and baktun (20 katun). So the full cycle repeats every  $20 \times 18 \times 20 \times 20 = 144,000$  days (approximately 394 years).

**Value**

A mayan vector object

**See Also**

[cal\\_mayan](#)

**Examples**

```
gregorian_date(2012, 12, 10:30) |>
  as_mayan()
```

---

new\_calendar

*Define calendar objects*

---

**Description**

Generate a calendar object of class "calendar". Examples of calendars produced in this way include cal\_chinese, cal\_gregorian, cal\_hebrew, cal\_islamic, and cal\_iso.

**Usage**

```
new_calendar(  
  name,  
  short_name,  
  granularities,  
  validate_granularities,  
  format,  
  from_rd,  
  to_rd  
)
```

cal\_babylonian

cal\_bahai

cal\_balinese

cal\_chinese

cal\_japanese

cal\_korean

cal\_vietnamese

cal\_coptic

cal\_ethiopic

cal\_egyptian

cal\_armenian

cal\_french

cal\_afrench

cal\_gregorian

cal\_hebrew

cal\_icelandic

cal\_islamic

cal\_iso

cal\_julian

cal\_oislamic

cal\_saudi

cal\_ohebrew

cal\_samaritan

cal\_mayan

cal\_hindu\_lunar

cal\_hindu\_solar

cal\_old\_hindu\_solar

cal\_old\_hindu\_lunar

cal\_persian

cal\_apersian

cal\_roman

cal\_tibetan

### Arguments

name	Name of calendar
short_name	Short name of calendar
granularities	Character vector with names of granularities of calendar (e.g., for the Gregorian calendar, the granularities are year, month, and day).
validate_granularities	Function to check granularities are valid (e.g., Gregorian months should be between 1 and 12).
format	Function to specify date format as a character string.
from_rd	Function to convert from RD to calendar date.
to_rd	Function to convert from calendar date to RD.

### Value

A calendar object of class "calendar"

### Examples

```
cal_gregorian
tibble::tibble(
  x = new_date(year = 2025, month = 5, day = 1:31, calendar = cal_gregorian),
  y = as_date(x, calendar = cal_islamic)
)
```

---

new\_moons

*Full moons and new moons in Gregorian years*

---

### Description

Calculate all the near-full or near-new moons in a vector of Gregorian years

### Usage

```
new_moons(year)
```

```
full_moons(year)
```

**Arguments**

year                    A vector of Gregorian years

**Value**

A vector of Gregorian dates representing the full moons or new moons in the given years.

A vector of dates

**Examples**

```
full_moons(2025)
new_moons(2025)
```

---

persian_date	<i>Persian dates</i>
--------------	----------------------

---

**Description**

The modern Persian calendar was adopted in 1925 in Iran and in 1957 in Afghanistan. An alternative version of the calendar, using only arithmetic (rather than astronomical) calculations is available as the *apersian* calendar.

**Usage**

```
persian_date(year = integer(), month = integer(), day = integer())
```

```
apersian_date(year = integer(), month = integer(), day = integer())
```

```
as_persian(date)
```

```
as_apersian(date)
```

**Arguments**

year                    Numeric vector of years  
month                    Numeric vector of months  
day                      Numeric vector of days  
date                     Vector of dates on some calendar

**Value**

A persian vector object

**Examples**

```
gregorian_date(2025, 5, 1:20) |>
  as_persian()
```

---

roman_date	<i>Roman calendar dates</i>
------------	-----------------------------

---

### Description

The Roman calendar (as defined here) is the same as the Julian calendar but with different nomenclature. Rather than use a (year, month, day) triple for each date, it specifies dates using year, month, event, count.

### Usage

```
roman_date(
  year = integer(),
  month = integer(),
  event = integer(),
  count = integer(),
  leap_day = logical()
)
```

```
as_roman(date)
```

### Arguments

year	A numeric vector of years
month	A numeric vector of months
event	A numeric vector of events: 1 = Kalends, 2 = Nones, 3 = Ides
count	A numeric vector of counts
leap_day	A logical vector indicating if day is a leap day
date	Vector of dates on some calendar

### Value

A roman vector object

### See Also

[cal\\_roman](#)

### Examples

```
roman_date(66, 4, 1, 1, FALSE)
new_date(year = 66, month = 4, event = 1, count = 1, leap_day = FALSE, calendar = cal_roman)
as_roman("2016-01-01")
tibble::tibble(
  x = seq(as.Date("2025-01-01"), as.Date("2025-12-31"), by = "day"),
  y = as_roman(x)
)
```

---

sunrise	<i>Sun and moon rise and set given a date and location</i>
---------	--

---

### Description

Calculate the time of sunrise, sunset, moonrise and moonset at a specific location and date. The time zone of the location is used as specified in the location object. No adjustments are made for daylight saving.

### Usage

```
sunrise(date, location)
```

```
sunset(date, location)
```

```
moonset(date, location)
```

```
moonrise(date, location)
```

### Arguments

date	Vector of dates on some calendar.
location	Vector of locations of class "location", usually the output from the location function

### Value

Time of sunrise

### Examples

```
melbourne <- location(-37.8136, 144.9631, 31, 10)
sydney <- location(-33.8688, 151.2093, 3, 10)
sunrise(gregorian_date(2025, 1, 1), c(melbourne, sydney))
sunset(gregorian_date(2025, 1, 1), c(melbourne, sydney))
moonrise(gregorian_date(2025, 1, 1), c(melbourne, sydney))
moonset(gregorian_date(2025, 1, 1), c(melbourne, sydney))
```

---

tibetan_date	<i>Tibetan calendar dates</i>
--------------	-------------------------------

---

### Description

There are several Tibetan calendars. These functions implement the official Phuglugs version of the Kalachakra calendar, which is similar to the Hindu lunisolar calendars.

**Usage**

```
tibetan_date(  
  year = integer(),  
  month = integer(),  
  leap_month = logical(),  
  day = integer(),  
  leap_day = logical()  
)  
  
as_tibetan(date)
```

**Arguments**

year	A numeric vector of years
month	A numeric vector of months
leap_month	A logical vector of leap months
day	A numeric vector of days
leap_day	A logical vector of leap days
date	A vector of dates on some calendar

**Value**

A tibetan\_date object

**See Also**

[tibetan\\_new\\_year](#)

**Examples**

```
gregorian_date(2025, 6, 1:10) |> as_tibetan()
```

---

tibetan_new_year	<i>Tibetan holidays</i>
------------------	-------------------------

---

**Description**

The Tibetan New Year occurs on the first day of the Tibetan calendar. These functions calculate the date given either a Gregorian year or a Tibetan year. Both return a Gregorian date.

**Usage**

```
tibetan_new_year(year)  
  
losar(t_year)
```

**Arguments**

year            A vector of Gregorian years  
t\_year         A vector of Tibetan years

**Value**

A vector of Gregorian dates corresponding to the Tibetan New Year

**See Also**

[tibetan\\_date](#)

**Examples**

```
tibetan_new_year(2025:2028)  
losar(2152:2154)
```

---

time\_of\_day

*Time of day*

---

**Description**

Create a time object

**Usage**

```
time_of_day(hour = integer(), minute = integer(), second = numeric())
```

**Arguments**

hour            A numeric vector of hours  
minute         A numeric vector of minutes  
second         A numeric vector of seconds

**Value**

A time\_of\_day vector object, stored as a vctrs record containing hours, minutes and seconds.

---

us_memorial_day	<i>US Holidays</i>
-----------------	--------------------

---

**Description**

Functions to return Gregorian dates for US holidays and other special days

**Usage**

```
us_memorial_day(year)
us_independence_day(year)
us_labor_day(year)
us_election_day(year)
us_daylight_saving_start(year)
us_daylight_saving_end(year)
unlucky_fridays(year)
```

**Arguments**

year	Gregorian year
------	----------------

**Value**

A vector of Gregorian dates corresponding to the US holidays or special days.

**Examples**

```
us_memorial_day(2025)
us_independence_day(2025)
us_labor_day(2025)
us_election_day(2025)
us_daylight_saving_start(2025)
us_daylight_saving_end(2025)
unlucky_fridays(2025)
```

---

`yom_kippur`*Jewish Holidays*

---

**Description**

Functions to return Gregorian dates for various Jewish holidays

**Usage**`yom_kippur(year)``passover(year)``purim(year)``ta_anit_esther(year)``tishah_be_av(year)``hanukkah(year)``rosh_hashanah(year)``sukkot(year)``shavuot(year)`**Arguments**

`year`            A numeric vector of Gregorian years

**Value**

A vector of dates on the Gregorian calendar

**See Also**

[hebrew\\_date](#)

**Examples**

```
tibble::tibble(
  year = 2025:2030,
  ta_anit_esther = ta_anit_esther(year),
  purim = purim(year),
  passover = passover(year),
  shavuot = shavuot(year),
  tishah_be_av = tishah_be_av(year),
```

*yom\_kippur*

39

```
rosh_hashanah = rosh_hashanah(year),  
yom_kippur = yom_kippur(year),  
sukkot = sukkot(year),  
hanukkah = hanukkah(year)  
)
```

# Index

## \* datasets

- new\_calendar, 29
- advent, 3
- afrench\_date (french\_date), 15
- apersian\_date (persian\_date), 32
- armenian\_date (egyptian\_date), 14
- as\_afrench (french\_date), 15
- as\_apersian (persian\_date), 32
- as\_armenian (egyptian\_date), 14
- as\_babylonian (babylonian\_date), 5
- as\_bahai (bahai\_date), 6
- as\_balinese (balinese\_date), 8
- as\_chinese (chinese\_date), 9
- as\_coptic (coptic\_date), 11
- as\_date, 4
- as\_egyptian (egyptian\_date), 14
- as\_ethiopic (coptic\_date), 11
- as\_french (french\_date), 15
- as\_gregorian (gregorian\_date), 17
- as\_hebrew (hebrew\_date), 18
- as\_hindu\_lunar (hindu\_solar\_date), 20
- as\_hindu\_solar (hindu\_solar\_date), 20
- as\_icelandic (icelandic\_date), 21
- as\_islamic (islamic\_date), 22
- as\_iso (iso\_date), 24
- as\_japanese (chinese\_date), 9
- as\_julian (julian\_date), 25
- as\_korean (chinese\_date), 9
- as\_mayan (mayan\_date), 28
- as\_ohebrew (hebrew\_date), 18
- as\_oislamic (islamic\_date), 22
- as\_old\_hindu\_lunar (hindu\_solar\_date), 20
- as\_old\_hindu\_solar (hindu\_solar\_date), 20
- as\_persian (persian\_date), 32
- as\_roman (roman\_date), 33
- as\_samaritan (hebrew\_date), 18
- as\_saudi (islamic\_date), 22
- as\_tibetan (tibetan\_date), 34
- as\_time\_of\_day, 4
- as\_vietnamese (chinese\_date), 9
- astronomical\_easter (advent), 3
- babylonian\_date, 5
- bahai\_date, 6, 7
- bahai\_new\_year, 7, 7
- balinese\_date, 8, 26
- birth\_of\_the\_bab (bahai\_new\_year), 7
- cal\_afrench (new\_calendar), 29
- cal\_apersian (new\_calendar), 29
- cal\_armenian (new\_calendar), 29
- cal\_babylonian, 6
- cal\_babylonian (new\_calendar), 29
- cal\_bahai, 7
- cal\_bahai (new\_calendar), 29
- cal\_balinese (new\_calendar), 29
- cal\_chinese, 10
- cal\_chinese (new\_calendar), 29
- cal\_coptic, 12
- cal\_coptic (new\_calendar), 29
- cal\_egyptian (new\_calendar), 29
- cal\_ethiopic, 12
- cal\_ethiopic (new\_calendar), 29
- cal\_french (new\_calendar), 29
- cal\_gregorian, 17
- cal\_gregorian (new\_calendar), 29
- cal\_hebrew, 18
- cal\_hebrew (new\_calendar), 29
- cal\_hindu\_lunar, 21
- cal\_hindu\_lunar (new\_calendar), 29
- cal\_hindu\_solar, 21
- cal\_hindu\_solar (new\_calendar), 29
- cal\_icelandic (new\_calendar), 29
- cal\_islamic, 23
- cal\_islamic (new\_calendar), 29
- cal\_iso, 25
- cal\_iso (new\_calendar), 29

- cal\_japanese (new\_calendar), 29
- cal\_julian, 25
- cal\_julian (new\_calendar), 29
- cal\_korean (new\_calendar), 29
- cal\_mayan, 29
- cal\_mayan (new\_calendar), 29
- cal\_ohebrew (new\_calendar), 29
- cal\_oislamic (new\_calendar), 29
- cal\_old\_hindu\_lunar, 21
- cal\_old\_hindu\_lunar (new\_calendar), 29
- cal\_old\_hindu\_solar, 21
- cal\_old\_hindu\_solar (new\_calendar), 29
- cal\_persian (new\_calendar), 29
- cal\_roman, 33
- cal\_roman (new\_calendar), 29
- cal\_samaritan (new\_calendar), 29
- cal\_saudi (new\_calendar), 29
- cal\_tibetan (new\_calendar), 29
- cal\_vietnamese (new\_calendar), 29
- chinese\_date, 9, 11
- chinese\_new\_year, 10, 11
- christmas (advent), 3
- coptic\_christmas, 12
- coptic\_christmas (advent), 3
- coptic\_date, 11
- day\_of\_month (day\_of\_week), 12
- day\_of\_week, 12
- day\_of\_year (day\_of\_week), 12
- days\_remaining (day\_of\_week), 12
- diwali, 21
- diwali (hindu\_lunar\_new\_year), 19
- dragon\_festival (chinese\_new\_year), 11
- easter (advent), 3
- egyptian\_date, 14
- eid\_al\_adha (islamic\_new\_year), 23
- eid\_al\_fitr (islamic\_new\_year), 23
- epiphany (advent), 3
- ethiopic\_date (coptic\_date), 11
- feast\_of\_ridvan (bahai\_new\_year), 7
- french\_date, 15
- full\_moons (new\_moons), 31
- granularity (granularity\_names), 16
- granularity\_names, 16
- gregorian\_date, 17
- hanukkah (yom\_kippur), 38
- hebrew\_date, 18, 38
- hindu\_lunar\_date, 20
- hindu\_lunar\_date (hindu\_solar\_date), 20
- hindu\_lunar\_new\_year, 19
- hindu\_solar\_date, 20
- icelandic\_date, 21
- islamic\_date, 22, 24
- islamic\_new\_year, 23
- iso\_date, 24
- japanese\_date (chinese\_date), 9
- julian\_date, 25
- kajeng\_keliwon, 9, 26
- korean\_date (chinese\_date), 9
- location, 27
- losar (tibetan\_new\_year), 35
- lunar\_phase, 27
- mawlid (islamic\_new\_year), 23
- mayan\_date, 28
- mesha\_sankranti (hindu\_lunar\_new\_year), 19
- month\_of\_year (day\_of\_week), 12
- moonrise (sunrise), 34
- moonset (sunrise), 34
- naw\_ruz (bahai\_new\_year), 7
- new\_calendar, 29
- new\_date (as\_date), 4
- new\_moons, 31
- ohebrew\_date (hebrew\_date), 18
- oislamic\_date (islamic\_date), 22
- old\_hindu\_lunar\_date (hindu\_solar\_date), 20
- old\_hindu\_solar\_date (hindu\_solar\_date), 20
- orthodox\_christmas (advent), 3
- orthodox\_easter (advent), 3
- passover (yom\_kippur), 38
- pentecost (advent), 3
- persian\_date, 32
- purim (yom\_kippur), 38
- qing\_ming (chinese\_new\_year), 11
- rama (hindu\_lunar\_new\_year), 19

ramadan, 23  
ramadan (islamic\_new\_year), 23  
roman\_date, 33  
rosh\_hashanah, 18  
rosh\_hashanah (yom\_kippur), 38

sacred\_wednesdays  
    (hindu\_lunar\_new\_year), 19  
samaritan\_date (hebrew\_date), 18  
saudi\_date (islamic\_date), 22  
shavuot (yom\_kippur), 38  
shiva (hindu\_lunar\_new\_year), 19  
sukkot (yom\_kippur), 38  
sunrise, 34  
sunset (sunrise), 34

ta\_anit\_esther (yom\_kippur), 38  
tibetan\_date, 34, 36  
tibetan\_new\_year, 35, 35  
time\_of\_day, 5, 36  
tishah\_be\_av (yom\_kippur), 38  
tumpek (kajeng\_keliwon), 26

unlucky\_fridays (us\_memorial\_day), 37  
us\_daylight\_saving\_end  
    (us\_memorial\_day), 37  
us\_daylight\_saving\_start  
    (us\_memorial\_day), 37  
us\_election\_day (us\_memorial\_day), 37  
us\_independence\_day (us\_memorial\_day),  
    37  
us\_labor\_day (us\_memorial\_day), 37  
us\_memorial\_day, 37

vietnamese\_date (chinese\_date), 9

week\_of\_month (day\_of\_week), 12  
week\_of\_year, 16, 24, 25  
week\_of\_year (day\_of\_week), 12

year (day\_of\_week), 12  
yom\_kippur, 38