Title

stata.com

Datetime relative dates — Obtaining dates and date information from other dates

Description Quick start Syntax Remarks and examples Also see

Description

This entry describes functions that calculate dates from other dates, such as the date of a birthday in another year or the next leap year after a given year. It also describes functions that return the current date and current datetime.

Quick start

Display today's date

```
display %td today()
```

Save the current date and time in a scalar

```
scalar ctime = now()
```

Calculate the date of a birthday in the year given by numeric variable `y` based on a numerically encoded Stata date variable `dob` that gives date of birth

```
generate bday_future = birthday(dob, y)
```

Same as above, but for persons born on 29feb have their birthdays on 28feb in nonleap years (rather than the default of 01mar)

```
generate bday_future = birthday(dob, y, "28feb")
```

Calculate the date of the first birthday after Stata date `date_today` based on date of birth `dob`

```
generate next_bday = nextbirthday(dob, date_today)
```

Calculate the number of days in the year `y`

```
generate ndays = cond(isleapyear(y), 366, 365)
```

Calculate the year of the leap year immediately before the year `y`

```
generate yleap = previousleapyear(y)
```

Calculate the number of days in the month on which the values of Stata date variable `d` fall

```
generate ndays = daysinmonth(d)
```

Calculate the date of the first Friday of month `m` and year `y`

```
generate firstfriday = firstweekdayofmonth(m, y, "Friday")
```

Calculate the date of the previous Saturday relative to Stata date `d`

```
generate previous = previousweekday(d, "sat")
```

Syntax

Description	Function	Value returned
today	`today()`	Stata date
current date and time	`now()`	Stata datetime/c
birthday in year	`birthday(`$e_{d\,\mathrm{DOB}}$`,`Y`[,`s_{nl}`])`	Stata date
previous birthday	`previousbirthday(`$e_{d\,\mathrm{DOB}}$`,`e_d`[,`s_{nl}`])`	Stata date
next birthday	`nextbirthday(`$e_{d\,\mathrm{DOB}}$`,`e_d`[,`s_{nl}`])`	Stata date
days in month	`daysinmonth(`e_d`)`	28–31
first day of month	`firstdayofmonth(`e_d`)`	Stata date
last day of month	`lastdayofmonth(`e_d`)`	Stata date
leap year indicator	`isleapyear(`Y`)`	0 or 1
previous leap year	`previousleapyear(`Y`)`	year
next leap year	`nextleapyear(`Y`)`	year
leap second indicator	`isleapsecond(`e_{tC}`)`	0 or 1
first day of week of month	`firstweekdayofmonth(`M`,`Y`,`d`)` or `firstdowofmonth(`M`,`Y`,`d`)`	Stata date
last day of week of month	`lastweekdayofmonth(`M`,`Y`,`d`)` or `lastdowofmonth(`M`,`Y`,`d`)`	Stata date
previous day of week	`previousweekday(`e_d`,`d`)` or `previousdow(`e_d`,`d`)`	Stata date
next day of week	`nextweekday(`e_d`,`d`)` or `nextdow(`e_d`,`d`)`	Stata date

e_d and $e_{d\,\mathrm{DOB}}$ are Stata dates.
e_{tC} is a Stata datetime/C value (UTC time with leap seconds).
s_{nl} is a string specifying nonleap-year birthdays of 29feb and may be
`"01mar"`, `"1mar"`, `"mar01"`, or `"mar1"` (the default); or
`"28feb"` or `"feb28"` (case insensitive).
Y is a numeric year.
d is a numeric day of week (`0`=Sunday, `1`=Monday, ..., `6`=Saturday); alternatively,
it is a string specifying the first two or more letters of the day of week (case insensitive).

Note: The string s_{nl} specifying nonleap-year birthdays is an optional argument. It rarely needs to be specified. See example 3 in [D] **Datetime durations**.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Current date and time
Birthdays and anniversaries
Months: Number of days, first day, and last day
Determining leap years
Determining leap seconds
Dates of days of week

We assume you have read [D] **Datetime** and are familiar with how Stata stores and formats dates.

Current date and time

`today()` and `now()` return date and datetime/c values for today's date and the current datetime, respectively. Note that the datetime value returned by `now()` is not adjusted for leap seconds.

Birthdays and anniversaries

The `birthday()` function returns a Stata date giving the birthday in a specified year. For example, suppose `date_of_birth` is a variable containing Stata dates and `yvar` is a numeric variable containing years; typing

```
. generate bday = birthday(date_of_birth, yvar)
```

produces a Stata date variable `bday` containing birthdays in those years. However, it will not be formatted as a date variable. If you list `bday`, you will see numbers, not dates. To see dates, you must give it a date format, such as

```
. format bday %td
```

We used the format `%td`, the simplest format specification for daily dates.

Of course, `birthday()` can be used for more than just birthdays. It can be used to give anniversary dates of any date in different years.

The `previousbirthday()` and `nextbirthday()` functions do what their names suggest. Typing

```
. generate pbday = previousbirthday(date_of_birth, current_date)
. format pbday %td
```

gives birthdays immediately before `current_date`. Typing

```
. generate nbday = nextbirthday(date_of_birth, current_date)
. format nbday %td
```

gives birthdays immediately after `current_date`. Note that if `current_date` is a birthday, `previousbirthday()` returns the previous birthday, not the value of `current_date`. Similarly, `nextbirthday()` returns the next birthday when the argument is a birthday.

The optional last argument, s_{nl}, for `birthday()`, `previousbirthday()`, and `nextbirthday()` applies only to a date of birth on 29feb. The argument controls whether to use 01mar (the default) or 28feb as the birthday in nonleap years. See example 3 in [D] **Datetime durations** and the example below.

Example 1: Birthdays in other years

Here we show how to use `birthday()` and `nextbirthday()` to calculate birthdays in other years. We load a dataset with Stata date variables `dob` and `date` and a numeric variable `year`.

```
. use https://www.stata-press.com/data/r18/birthdays
(Fictional data for calculating birthdays)
. list, sepby(dob)
```

	dob	date	year
1.	Mon 28 Aug 1967	Thu 27 Aug 2020	2020
2.	Mon 28 Aug 1967	Sat 28 Aug 2021	2021
3.	Mon 28 Aug 1967	Mon 29 Aug 2022	2022
4.	Thu 29 Feb 1968	Tue 28 Feb 2023	2023
5.	Thu 29 Feb 1968	Thu 29 Feb 2024	2024
6.	Thu 29 Feb 1968	Sat 01 Mar 2025	2025

To calculate the birthday in `year` based on date of birth `dob`, we type

```
. generate bday = birthday(dob, year)
. format bday %tdDay_DD_Mon_CCYY
. list dob year bday, sepby(dob)
```

	dob	year	bday
1.	Mon 28 Aug 1967	2020	Fri 28 Aug 2020
2.	Mon 28 Aug 1967	2021	Sat 28 Aug 2021
3.	Mon 28 Aug 1967	2022	Sun 28 Aug 2022
4.	Thu 29 Feb 1968	2023	Wed 01 Mar 2023
5.	Thu 29 Feb 1968	2024	Thu 29 Feb 2024
6.	Thu 29 Feb 1968	2025	Sat 01 Mar 2025

We see that for a date of birth of 28 Aug 1967, the birthday in 2020 is on 28 Aug 2020, which is a Friday. For persons born on leap day 29 Feb 1968, their birthdays in nonleap years will be on 01 Mar. In leap years, of course, they will be on 29 Feb.

Note that we used the fancy date format `%tdDay_DD_Mon_CCYY`. The `%td` at the beginning means it is a format for daily dates. `Day` displays the day of the week abbreviated. The underscore (_) means put in a space. `DD` displays the day with a leading zero. `Mon` displays the month abbreviated. `CCYY` displays the year with the century. See [D] **Datetime display formats** for all the format variants.

For persons born on leap days ("leaplings"), we can change the day of their birthdays in nonleap years from the default of 01 Mar to 28 Feb by specifying the optional argument `"28feb"`. For example,

```
. generate abday = birthday(dob, year, "28feb")
. format abday %tdDay_DD_Mon_CCYY
. list dob year abday, sepby(dob)
```

	dob	year	abday
1.	Mon 28 Aug 1967	2020	Fri 28 Aug 2020
2.	Mon 28 Aug 1967	2021	Sat 28 Aug 2021
3.	Mon 28 Aug 1967	2022	Sun 28 Aug 2022
4.	Thu 29 Feb 1968	2023	Tue 28 Feb 2023
5.	Thu 29 Feb 1968	2024	Thu 29 Feb 2024
6.	Thu 29 Feb 1968	2025	Fri 28 Feb 2025

Birthdays of leaplings are now on 28 Feb in nonleap years. Birthdays for nonleaplings are unaffected by this argument.

Suppose we want a birthday relative to another date. Say we want the date of the first birthday after `date`. We can do this by typing

```
. generate nbday = nextbirthday(dob, date)
. format nbday %tdDay_DD_Mon_CCYY
. list dob date nbday, sepby(dob)
```

	dob	date	nbday
1.	Mon 28 Aug 1967	Thu 27 Aug 2020	Fri 28 Aug 2020
2.	Mon 28 Aug 1967	Sat 28 Aug 2021	Sun 28 Aug 2022
3.	Mon 28 Aug 1967	Mon 29 Aug 2022	Mon 28 Aug 2023
4.	Thu 29 Feb 1968	Tue 28 Feb 2023	Wed 01 Mar 2023
5.	Thu 29 Feb 1968	Thu 29 Feb 2024	Sat 01 Mar 2025
6.	Thu 29 Feb 1968	Sat 01 Mar 2025	Sun 01 Mar 2026

We see that the first birthday after 27 Aug 2020 for someone born on 28 Aug is 28 Aug 2020. The first birthday after 28 Aug 2021 (a birthday) for someone born on 28 Aug is the birthday in the next year, 28 Aug 2022.

The first birthday after 29 Feb 2024 for someone born on 29 Feb is 01 Mar 2025. Again, we can specify the argument `"28feb"` to change the nonleap-year birthdays of leaplings to 28 Feb.

```
. generate anbday = nextbirthday(dob, date, "28feb")
. format anbday %tdDay_DD_Mon_CCYY
. list dob date anbday, sepby(dob)
```

	dob	date	anbday
1.	Mon 28 Aug 1967	Thu 27 Aug 2020	Fri 28 Aug 2020
2.	Mon 28 Aug 1967	Sat 28 Aug 2021	Sun 28 Aug 2022
3.	Mon 28 Aug 1967	Mon 29 Aug 2022	Mon 28 Aug 2023
4.	Thu 29 Feb 1968	Tue 28 Feb 2023	Thu 29 Feb 2024
5.	Thu 29 Feb 1968	Thu 29 Feb 2024	Fri 28 Feb 2025
6.	Thu 29 Feb 1968	Sat 01 Mar 2025	Sat 28 Feb 2026

Now the first birthday after 29 Feb 2024 for someone born on 29 Feb is 28 Feb 2025.

◁

Months: Number of days, first day, and last day

`daysinmonth(`e_d`)`, `firstdayofmonth(`e_d`)`, and `lastdayofmonth(`e_d`)` each take a Stata date e_d as an argument and determine the month of that date. `daysinmonth()` returns the number of days in that month. `firstdayofmonth()` returns the date of the first day of that month. `lastdayofmonth()` returns the date of the last day of that month.

For example, for any day in the month of February of leap year 2020 (such as 15feb2020), these functions return the following:

```
. display daysinmonth(mdy(2,15,2020))
29
. display %td firstdayofmonth(mdy(2,15,2020))
01feb2020
. display %td lastdayofmonth(mdy(2,15,2020))
29feb2020
```

Determining leap years

`isleapyear(`Y`)`, `previousleapyear(`Y`)`, and `nextleapyear(`Y`)` are functions that make it easier to handle leap years. Each takes a single argument that is a numeric year.

`isleapyear(`Y`)` returns 1 if Y is a leap year and 0 otherwise. The argument Y can be a numeric variable or a literal value. Here are some examples with literal values:

```
. display isleapyear(2020)
1
. display isleapyear(2021)
0
. display isleapyear(2100)
0
. display isleapyear(2400)
1
```

The year 2020 is a leap year, and 2021 is not. The year 2100 is not because it is divisible by 100 and not by 400. The year 2400 is divisible by 400, so it is a leap year.

`previousleapyear(`Y`)` returns the leap year immediately before year Y. `nextleapyear(`Y`)` returns the first leap year after year Y. Here are examples:

```
. display previousleapyear(2023)
2020
. display nextleapyear(2023)
2024
. display previousleapyear(2024)
2020
. display nextleapyear(2024)
2028
```

As you can see, when the argument is a leap year, these functions return the next leap year or previous leap year and not the leap year argument.

Determining leap seconds

`isleapsecond()` takes a datetime/C value (UTC time) as an argument and returns 1 (true) if that datetime is one of the 1,000 milliseconds of a leap second and 0 (false) otherwise. For example, the first leap second was introduced on 30jun1972, after the last millisecond of the day. Here is what `isleapsecond()` returns at various points in time, including right before the leap second was added on 30jun1972 (at 23:59.999) and right after the leap second was added on 01jul1972 (at 00:00.000). We use `tC()` to create datetime/C values.

```
. display isleapsecond(tC(30jun1972 23:59:59.999))
0
. display isleapsecond(tC(30jun1972 23:59:60.000))
1
. display isleapsecond(tC(30jun1972 23:59:60.999))
1
. display isleapsecond(tC(01jul1972 00:00:0))
0
```

`isleapsecond()` is useful for determining whether datetime/C values can be converted to datetime/c without any loss of information. Suppose we have a variable `admitTime` that contains times of patient admissions as datetime/C values. We can type the following:

```
. generate anyleapsec = isleapsecond(admitTime)
. tabulate anyleapsec
```

anyleapsec	Freq.	Percent	Cum.
0	1,064	100.00	100.00
Total	1,064	100.00	

`anyleapsec` is all zero, so no patient was admitted on a leap second, and we can convert `admitTime` to datetime/c without any times being altered.

```
. generate newTime = cofC(admitTime)
```

Had there been leap seconds in the data, `cofC()` would have converted the leap-second times to times one second later. For example,

```
. display %tc cofC(tC(31dec2016 23:59:60))
01jan2017 00:00:00
```

Dates of days of week

`firstweekdayofmonth(`M`,`Y`,`d`)` and `lastweekdayofmonth(`M`,`Y`,`d`)` return the Stata date of the first and last day-of-week d, respectively, in month M of year Y. For example, we can find the first Monday of January 2000 with the command

```
. display %td firstweekdayofmonth(1, 2000, "Monday")
03jan2000
```

`previousweekday(`e_d`,`d`)` returns the Stata date corresponding to the last day-of-week d before the Stata date e_d. `nextweekday(`e_d`,`d`)` returns the Stata date corresponding to the first day-of-week d after the Stata date e_d. For example, the date of the first Saturday after today can be found with the command

```
. display %td nextweekday(today(), "sat")
25mar2023
```

Note that day-of-week d can be specified as an integer (`0` = Sunday, `1` = Monday, . . . , `6` = Saturday) or as a string with the first two or more letters of the day of the week (case insensitive). For example, Sunday can be specified as `0` or `"Sunday"`, `"Sun"`, `"su"`, etc.

Also see

[D] **Datetime** — Date and time values and variables

[D] **Datetime business calendars** — Business calendars

[D] **Datetime conversion** — Converting strings to Stata dates

[D] **Datetime display formats** — Display formats for dates and times

[D] **Datetime durations** — Obtaining and working with durations

[D] **Datetime values from other software** — Date and time conversion from other software