RStudio® is a trademark of RStudio, Inc.  •  CC BY SA RStudio •  info@rstudio.com  •  844-448-1212 • rstudio.com •  Learn more at lubridate.tidyverse.org  •  lubridate 1.6.0  •   Updated: 2017-12

### Dates and times with lubridate :: CHEAT SHEET

#### Date-times

- **2017-11-28 12:00:00**
  - A date-time is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

#### Parse Date-times (Convert strings or numbers to date-times)

1. Identify the order of the year (y), month (m), day (d), hour (h), minute (m) and second (s) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

#### Get and Set Components

- Use an accessor function to get a component.
- Assign into an accessor function to change a component in place.

#### Round Date-times

- **floor_date(x, unit = "second")** Round down to nearest unit.
- **round_date(x, unit = "month")** Round to nearest unit.
- **ceiling_date(x, unit = "second")** Change on boundary = NULL) Round up to nearest unit.
- **rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE)** Roll back to last day of previous month. rollback(dt)

#### Stamp Date-times

- **stamp()** Derive a template from an example string and return a new function that will apply the template to date-times. Also `stamp_date()` and `stamp_time()`.
   1. Derive a template, create a function `sf <- stamp("Created Monday, Apr 05, 2010 00:00")`
   2. Apply the template to dates `sf(ymd("2010-04-05"))` 

#### Time Zones

- R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns one time zone per vector.
- Use the UTC time zone to avoid Daylight Savings.
- `OlsonNames()` Returns a list of valid time zone names. `OlsonNames()`
Math with Date-times – Lubridate provides three classes of timespans to facilitate math with dates and date-times

**PERIODS**
Add or subtract periods to model events that happen at specific clock times, like the NYSE opening bell.

Make a period with the name of a time unit **pluralized**, e.g.

```r
p = months(3) + days(12)
```

**DURATIONS**
Add or subtract durations to model physical processes, like battery life. Durations are stored as seconds, the only time unit with a consistent length. **Difftimes** are a class of durations found in base R.

Make a duration with the name of a period prefixed with a `d`, e.g.

```r
d = ddays(14)
d "1209600s (~2 weeks)"
```

**INTERVALLS**
Divide an interval by a duration to determine its physical length, divide an interval by a period to determine its implied length in clock time.

Make an interval with an interval() or `%>%`, e.g.

```r
i <- interval(ymd("2017-01-01"), d)
```

Not all years are 365 days due to **leap days**.
Not all minutes are 60 seconds due to **leap seconds**.

It is possible to create an imaginary date by adding months, e.g. February 31st `jan31 <- ymd(20180313)`
`jan31 + months(1)`
`# NA`

%m% and %m% will roll imaginary dates to the last day of the previous units.

`jan31 %m% months(1)`
`# "2018-02-28`

`add_with_rollback(e1, e2, roll_to_first = TRUE)` will roll imaginary dates to the first day of the new month.

`add_with_rollback(jan31, months(1), roll_to_first = TRUE)`
`# "2018-03-01"`

### PERIODS

A normal day
```r
nor <- ymd_hms("2018-01-01 01:30:00" ,tz="US/Eastern")
```

The start of daylight savings (spring forward)
```r
gap <- ymd_hms("2018-03-11 01:30:00" ,tz="US/Eastern")
```

The end of daylight savings (fall back)
```r
lap <- ymd_hms("2018-11-04 01:30:00" ,tz="US/Eastern")
```

Leap years and leap seconds
```r
leap <- ymd("2019-03-01")
```

### DURATIONS

**Number of days**
```r
dd <- ddays(14)
```

**Exact length in seconds**
```r
dyears(x = 1) 31536000x seconds.
dweeks(x = 1) 604800x seconds.
days(x = 1) 86400x seconds.
hours(x = 1) 3600x seconds.
minutes(x = 1) 60x seconds.
```

**Equivalent in common units**
```r
dminutes(x = 1) x minutes.
dseconds(x = 1) x seconds.
milliseconds(x = 1) x milliseconds.
microseconds(x = 1) x microseconds.
nanoseconds(x = 1) x nanoseconds.
picoseconds(x = 1) x picoseconds.
```

**period**(num = NULL, units = "second", …)
An automation friendly period constructor.
```r
period(5, unit = "years")
```

```r
as.period(x, unit) Coerce a timespan to a period, optionally in the specified units. Also is.period(). as.period()
```

**period_to_seconds**(x) Convert a period to the “standard” number of seconds implied by the period. Also seconds_to_period().

```r
period_to_seconds(p)
```

**LEAP YEARS**
```r
nor + months(1)
```

**INTERVALS**

Divide an interval by a duration to determine its physical length, divide an interval by a period to determine its implied length in clock time.

Make an interval with an interval() or `%>%`, e.g.

```r
j <- interval(ymd("2017-01-01"), d)
```

Not all years are 365 days due to **leap days**.
Not all minutes are 60 seconds due to **leap seconds**.

It is possible to create an imaginary date by adding months, e.g. February 31st `jan31 <- ymd(20180313)`
`jan31 + months(1)`
`# NA`

%m% and %m% will roll imaginary dates to the last day of the previous units.

`jan31 %m% months(1)`
`# "2018-02-28`

`add_with_rollback(e1, e2, roll_to_first = TRUE)` will roll imaginary dates to the first day of the new month.

`add_with_rollback(jan31, months(1), roll_to_first = TRUE)`
`# "2018-03-01"`

### INTERVALS

Divide an interval by a duration to determine its physical length, divide an interval by a period to determine its implied length in clock time.

Make an interval with an interval() or `%>%`, e.g.

```r
i <- interval(ymd("2017-01-01"), d)
```

Not all years are 365 days due to **leap days**.
Not all minutes are 60 seconds due to **leap seconds**.

It is possible to create an imaginary date by adding months, e.g. February 31st `jan31 <- ymd(20180313)`
`jan31 + months(1)`
`# NA`

%m% and %m% will roll imaginary dates to the last day of the previous units.

`jan31 %m% months(1)`
`# "2018-02-28`

`add_with_rollback(e1, e2, roll_to_first = TRUE)` will roll imaginary dates to the first day of the new month.

`add_with_rollback(jan31, months(1), roll_to_first = TRUE)`
`# "2018-03-01"`