Dates and times with lubridate :: CHEATSHEET

**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 tz argument to set the time zone, e.g. `ymd(x, tz = "UTC")`.

   - `ymd_hms()`, `ymd_hm()`, `ymd_h()`: `ymd_hms("2017-11-28T14:02:00")`
   - `ymd_hms()`, `ymd_hm()`, `ymd_h()`: `ymd_hms("2017-11-28 12:00:00")`
   - `mdy_hms()`, `mdy_hm()`, `mdy_h()`: `mdy_hms("July 4th, 2000")`
   - `dmy_hms()`, `dmy_hm()`, `dmy_h()`: `dmy_hms("4th of July '99")`
   - `dmy()`, `ymd()`, `my()`: `my("07-2020")`
   - `hms_hms()`, `hms_hm()`, `hms_h()`: `hms_hms("08:01:01")`

**ROUND DATE-TIMES**

- `floor_date(x, unit = "second")`: Round down to nearest unit. `floor_date(dt, unit = "month")`
- `round_date(x, unit = "second")`: Round to nearest unit. `round_date(dt, unit = "month")`
- `ceiling_date(x, unit = "second")`: Round up to nearest unit. `ceiling_date(dt, unit = "month")`
- `rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE)`: Rollback to last day of previous month. Also `rollforward()`, `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()`, `stamp_time()`

**TIME ZONES**

- `Sys.timezone()`: Gets current time zone.
- `OlsonNames()`: Returns a list of valid time zone names. OlsonNames()
- `with_tz(time, tzzone = "")`: Get the same date-time in a new time zone (a new clock time). Also `local_time(dt, tz, units)`, `with_tz(dt, "US/Pacific")`
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.

- years(x = 1) x years.
- months(x = 1) x months.
- days(x = 1) x days.
- minutes(x = 1) x minutes.
- seconds(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.
- as.period(x, unit) Coerce a timespan to a period, optionally in the specified units. Also is.period(). as.period(p).
- period_to_seconds(x) Convert a period to the "standard" number of seconds implied by the period. Also seconds_to_period(). period_to_seconds(p).

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.

Durations are a class of durations found in base R.

- dyears(x = 1) 31536000x seconds.
- dmonths(x = 1) 2629800x seconds.
- dweeks(x = 1) 604800x seconds.
- ddays(x = 1) 86400x seconds.
- dhours(x = 1) 3600x seconds.
- dminutes(x = 1) 60x seconds.
- dseconds(x = 1) 1x seconds.
- dmilliseconds(x = 1) 1x 10-3 seconds.
- dmilliseconds(x = 1) 1x 10-6 seconds.
- dnanoseconds(x = 1) 1x 10-9 seconds.
- dpicoseconds(x = 1) 1x 10-12 seconds.

duration(num = NULL, units = "second", ...) An automation friendly duration constructor. duration(s, unit = "years")

as.duration(x, ...) Coerce a timespan to a duration. Also is.duration(), is.difftime(). as.duration()

make_difftime(x) Make difftime with the specified number of units. make_difftime(9999)

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 interval() or %--%, e.g.

- i <- interval(ymd("2017-01-01"), d) i <- d %--% ymd("2017-12-31")
- j <- d %--% ymd("2017-01-01")

- a %within% b Does interval or date-time a fall within interval b? now() %within% 2017-01-01 2017-12-31
- int_start() Access/set the start date-time of an interval. Also int_end(). int_start(i) <- now(); int_end(i)
- int_aligns(i1, i2) Do two intervals share a boundary? Also int_overlaps(i1, i2)
- int_diff(times) Make the intervals that occur between the date-times in a vector.
- int_length(i) In seconds. int_length(i)
- int_shift(i, by) Shifts an interval up or down the timeline by a timespan. int_shift(i, days(1))
- as.interval(x, start, ...) Coerce a timespan to an interval with the start date-time. Also is.interval(). as.interval(days(1), start now)

Not all years are 365 days due to leap days.
Not all months are 30 days due to leap seconds.
It is possible to create an imaginary date by adding months, e.g. February 31st.
Not all years are 365 days due to leap seconds.