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

---

### Dates and Times

**Date-times**

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-11-28 12:00:00</td>
<td></td>
</tr>
</tbody>
</table>

**GET AND SET COMPONENTS**

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-11-28</td>
<td>12:00:00</td>
</tr>
</tbody>
</table>

**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
   ```r
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
   # sf(d) = "Created Sunday, Jan 17, 1999 3:34"
   ``
2. Apply the template to dates
   ```r
   sf(ymd("2010-04-05"))
   # d = "Created Monday, Apr 05, 2010 00:00"
   ```

**Round Date-times**

**floor_date(x, unit = "second")** Round down to nearest unit.

**round_date(x, unit = "second")** Round to nearest unit.

**ceiling_date(x, unit = "second")** Round up to nearest unit.

**rollback()**

Valid units are second, minute, hour, day, week, month, bimonth, quarter, season, halfyear and year.

**Time Zones**

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for a time zone. 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()

**Sys.timezone()** Gets current time zone.

---

**CC BY SA Posit Software, PBC • info@posit.co • posit.co • Learn more at lubridate.tidyverse.org • lubridate 1.9.2 • Updated: 2023-05**
Math with Date-times

Lubridate provides three classes of timespans to facilitate math with dates and date-times.

**Math with Date-times relies on the timeline,** which behaves inconsistently. Consider how the timeline behaves during:

- A normal day
  ```
  nor <- ymd_hms("2018-01-01 01:30:00")
  ```

- The start of daylight savings (spring forward)
  ```
  gap <- ymd_hms("2018-03-11 01:30:00")
  ```

- The end of daylight savings (fall back)
  ```
  lap <- ymd_hms("2018-11-04 00:30:00")
  ```

Leap years and leap seconds

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

Math with date-times relies on the timeline. Bounded by start and end date-times.

- **Periods** track changes in clock times, which ignore time line irregularities.
  ```
  nor + minutes(90)
  ```

- **Durations** track the passage of physical time, which deviates from clock time when irregularities occur.
  ```
  nor + dminutes(90)
  ```

- **Intervals** represent specific intervals of the timeline, bounded by start and end date-times.
  ```
  interval(nor, nor + minutes(90))
  ```

**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**
  ```
  years(x = 1) x years.
  ```
- **months**
  ```
  months(x = 1) x months.
  ```
- **days**
  ```
  days(x = 1) x days.
  ```
- **hours**
  ```
  hours(x = 1) x hours.
  ```
- **minutes**
  ```
  minutes(x = 1) x minutes.
  ```
- **seconds**
  ```
  seconds(x = 1) x seconds.
  ```

- **milliseconds**
  ```
  milliseconds(x = 1) x milliseconds.
  ```
- **microseconds**
  ```
  microseconds(x = 1) x microseconds.
  ```
- **nanoseconds**
  ```
  nanoseconds(x = 1) x nanoseconds.
  ```
- **picoseconds**
  ```
  picoseconds(x = 1) x picoseconds.
  ```

- **period**
  ```
  period(num = NULL, units = "second", ...) # An automation friendly period constructor.
  ```

**As.period**

Coerce a timespan to a period, optionally in the specified units. Also **is.period()**.

- **period_to_seconds**
  ```
  period_to_seconds(x) Convert a period to the "standard" number of seconds implied by the period. Also seconds_to_period().
  ```

**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. **Diftimes** are a class of durations found in base R.

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

- **dyears**
  ```
  dyears(x = 1) 131536000x seconds.
  ```
- **dmonths**
  ```
  dmonths(x = 1) 26298000x seconds.
  ```
- **dweeks**
  ```
  dweeks(x = 1) 604800x seconds.
  ```
- **ddays**
  ```
  ddays(x = 1) 86400x seconds.
  ```
- **dhours**
  ```
  dhours(x = 1) 3600x seconds.
  ```
- **dminutes**
  ```
  dminutes(x = 1) 60x seconds.
  ```
- **dseconds**
  ```
  dseconds(x = 1) x seconds.
  ```
- **dmilliseconds**
  ```
  dmilliseconds(x = 1) x 10^3 seconds.
  ```
- **dmicroseconds**
  ```
  dmicroseconds(x = 1) x 10^6 seconds.
  ```
- **dnanoseconds**
  ```
  nanoseconds(x = 1) x 10^9 seconds.
  ```
- **dpicoseconds**
  ```
  picoseconds(x = 1) x 10^12 seconds.
  ```

- **duration**
  ```
  duration(num = NULL, units = "second", ...) # An automation friendly duration constructor.
  ```

**As.duration**

Coerce a timespan to a duration. Also **is.duration()**, **is_diftimes**.

- **make_diftimes**
  ```
  make_diftimes(x) Make diftimes with the specified number of units.
  ```

**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.,**

- A period
  ```
  j <- interval(ymd("2017-01-01"), d)
  ```
- A period
  ```
  i <- interval(ymd(2017-01-01), d)
  ```

- **%within%**
  ```
  a %within% b Does interval or date-time a fall within interval b? Now() %within% ymd(2018-01-01)
  ```

- **int_start**
  ```
  int_start(int) Access/set the start date-time of an interval. Also int_end().
  ```

- **int_aligns**
  ```
  int_aligns(i, j) Do two intervals share a boundary? Also int_overlaps().
  ```

- **int_diff(times)**
  ```
  int_diff(i, j) Make the intervals that occur between the date-times in a vector.
  ```

- **int_flip(int)**
  ```
  int_flip(int) Reverses the direction of an interval. Also int_standardize().
  ```

- **int_length(int)**
  ```
  int_length(i) Length in seconds.
  ```

- **int_shift(int, by)**
  ```
  int_shift(i, by) Shifts an interval up or down a number of units. int_shift(i, days(-1))
  ```

- **as.interval(x, start, ...) Coerce a timespan to an interval with the start date-time. Also is.interval().**

---

**CC BY-SA Posit Software, PBC • info@posit.co • posit.co • Learn more at lubridate.tidyverse.org • lubridate 1.9.2 • Updated: 2023-05**