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
  - `dt <- as_datetime(1511870400)  # "2017-11-28 12:00:00 UTC"

- **2017-11-28**
  - A date is a day stored as the number of days since 1970-01-01
  - `d <- as_date(17498)  # "2017-11-28"

- **12:00:00**
  - An hms is a time stored as the number of seconds since 00:00:00
  - `t <- hms::as.hms(85)  # 00:01:25`

### Parse Date-times

- **2017-11-28 T14:02:00**
- **2017-22-12 10:00:00**
- **"2016-01-01"**
- **January 2017**
- **2001**
- **July 2017**
- **2017**
- **1.28.2017**
- **2.28.2017**
- **3.31.2017**
- **4.4.2017**
- **5.5.2017**
- **6.6.2017**
- **7.7.2017**
- **8.8.2017**
- **9.9.2017**
- **10.10.2017**
- **11.11.2017**
- **12.12.2017**
- **13.13.2017**
- **15.15.2017**
- **16.16.2017**
- **17.17.2017**
- **18.18.2017**
- **19.19.2017**
- **20.20.2017**
- **21.21.2017**
- **22.22.2017**
- **23.23.2017**
- **24.24.2017**
- **25.25.2017**
- **26.26.2017**
- **27.27.2017**
- **28.28.2017**
- **29.29.2017**
- **30.30.2017**
- **31.31.2017**
- **2017-11-28 12:00:00**

### 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.
  - `floor_date(dt, unit = "month")`

- **round_date(x, unit = "second")**
  - Round to nearest unit.
  - `round_date(dt, unit = "month")`

- **ceiling_date(x, unit = "second")**, **change_on_boundary = NULL**
  - Round up to nearest unit.
  - `ceiling_date(dt, unit = "month")`

- **rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE)**
  - Roll back to last day of previous month.

### Stamp Date-times

- **stamp(x)**
  - 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 Sunday, Jan 17, 1999 3:34")`
  2. **Apply the template to dates** `sf(sdf("2010-04-05"))  # [1] "Created Monday, Apr 05, 2010 00:00"

### 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()**

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: 2017-12-01
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 00:30:00", tz="US/Eastern")`
- The start of daylight savings (spring forward): `gap <- ymd_hms("2018-03-01 01:30:00", tz="US/Eastern")`
- The end of daylight savings (fall back): `lap <- ymd_hms("2018-11-04 00:30:00", tz="US/Eastern")`
- Leap years and leap seconds: `leap <- ymd("2019-03-01")`

**Periods** track changes in clock times, which ignore time line irregularities.

- Make a period with the name of a time unit: `p <- months(3) + days(12)`
- Add or subtract periods to model events that happen at specific clock periods: `normal + dyears(1)`

**Durations** track the passage of physical time, which deviates from clock time when irregularities occur.

- Make a duration with the name of a period prefixed with a `d`, e.g.: `dd <- ddays(14)`
- Add or subtract durations to model physical processes, like battery life: `ddays(x = 1) x days.`

**Intervals** represent specific intervals of the timeline, bounded by start and end date-times.

- Make an interval with `interval()`: `i <- interval(ymd("2017-01-01"), d)`
- Divide an interval by a duration to determine its physical length, divide and interval by a period to determine its implied length in clock time: `j <- d %--% i`

---

**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** `plurализed`, e.g.: `p <- months(3) + days(12)`
- **3m 12d 0H 0M 0S**

**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.: `dd <- ddays(14)`
- **Exact length in seconds**
- **Equivalent in common units**

**INTERVALS**

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

- **Make an interval with `interval()` or `as.interval()`, e.g.:** `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)`
- **%m% and %m-% will give an imaginary date to the last day of the previous month:**
- **The start of daylight savings (spring forward):** `Jan31 + months(1)`
- **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)**

---

**LEAP YEARS AND LEAP SECONDS**

- **Leap years and leap seconds:** `leap <- ymd("2019-03-01")`
- **The start of daylight savings (spring forward):** `gap <- ymd_hms("2018-03-01 01:30:00", tz="US/Eastern")`
- **The end of daylight savings (fall back):** `lap <- ymd_hms("2018-11-04 00:30:00", tz="US/Eastern")`
- **Leap years and leap seconds:** `leap <- ymd("2019-03-01")`

---

**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** `plurализed`, e.g.: `p <- months(3) + days(12)`
- **3m 12d 0H 0M 0S**

**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.: `dd <- ddays(14)`
- **Exact length in seconds**
- **Equivalent in common units**

**INTERVALS**

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

- **Make an interval with `interval()` or `as.interval()`, e.g.:** `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)`
- **%m% and %m-% will give an imaginary date to the last day of the previous month:**
- **The start of daylight savings (spring forward):** `Jan31 + months(1)`
- **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)**

---

**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** `plurализed`, e.g.: `p <- months(3) + days(12)`
- **3m 12d 0H 0M 0S**

**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.: `dd <- ddays(14)`
- **Exact length in seconds**
- **Equivalent in common units**

**INTERVALS**

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

- **Make an interval with `interval()` or `as.interval()`, e.g.:** `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)`
- **%m% and %m-% will give an imaginary date to the last day of the previous month:**
- **The start of daylight savings (spring forward):** `Jan31 + months(1)`
- **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)**

---

**LEAP YEARS AND LEAP SECONDS**

- **Leap years and leap seconds:** `leap <- ymd("2019-03-01")`
- **The start of daylight savings (spring forward):** `gap <- ymd_hms("2018-03-01 01:30:00", tz="US/Eastern")`
- **The end of daylight savings (fall back):** `lap <- ymd_hms("2018-11-04 00:30:00", tz="US/Eastern")`
- **Leap years and leap seconds:** `leap <- ymd("2019-03-01")`

---

**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** `plurализed`, e.g.: `p <- months(3) + days(12)`
- **3m 12d 0H 0M 0S**

**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.: `dd <- ddays(14)`
- **Exact length in seconds**
- **Equivalent in common units**

**INTERVALS**

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

- **Make an interval with `interval()` or `as.interval()`, e.g.:** `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)`
- **%m% and %m-% will give an imaginary date to the last day of the previous month:**
- **The start of daylight savings (spring forward):** `Jan31 + months(1)`
- **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)**

---

**LEAP YEARS AND LEAP SECONDS**

- **Leap years and leap seconds:** `leap <- ymd("2019-03-01")`
- **The start of daylight savings (spring forward):** `gap <- ymd_hms("2018-03-01 01:30:00", tz="US/Eastern")`
- **The end of daylight savings (fall back):** `lap <- ymd_hms("2018-11-04 00:30:00", tz="US/Eastern")`
- **Leap years and leap seconds:** `leap <- ymd("2019-03-01")`