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

#### Date-times

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

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

```r
dt <- as_datetime(2017-11-28 12:00:00)
d <- as_date(2017-11-28)
t <- hms(as.hms(85))
```

#### 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) in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

```r
ymd_hms(), ymd_hm(), ymd_h().
ymd_hms("2017-11-28T14:02:00")
```

#### GET AND SET COMPONENTS

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

```r
date(x) Date component. date(dt)
year(x) Year. year(dt)
isoyear(x) The ISO 8601 year.
epiyear(x) Epidemiological year.
month(x, label, abbr) Month. month(dt)
day(x) Day of month. day(dt)
wday(x, label, abbr) Day of week.
qday(x) Day of quarter.
hour(x) Hour. hour(dt)
minute(x) Minutes. minute(dt)
second(x) Seconds. second(dt)
week(x) Week of the year. week(dt)
iso_week(x) ISO 8601 week.
epweek(x) Epidemiological week.
quarter(x, with_year = FALSE) Quarter. quarter(dt)
semester(x, with_year = FALSE) Semester. semester(dt)
am(x) Is it in the am? am(dt)
pm(x) Is it in the pm? pm(dt)
dst(x) Is it daylight savings? dst(dt)
leap_year(x) Is it a leap year? leap_year(d)
```

#### Stamp Date-times

```r
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.
2. Apply the template to dates.

```r
sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
sf(ymd("2010-04-05"))
```

Tip: use a date with day > 12

#### Round Date-times

```r
floor_date(x, unit = "second") Round down to nearest unit.
round_date(dt, unit = "month")
```

```r
ceiling_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit.
```

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

#### 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.

```r
 OlsonNames() Returns a list of valid time zone names.
```

```r
with_tz(time, tz = "") Get the same date-time in a new time zone (a new clock time).
```
Math with Date-times

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

- **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 00:30:00", tz="US/Eastern")
  ```

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

- **nor + minutes(90)**
- **gap + minutes(90)**
- **lap + minutes(90)**

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

- **nor + dminutes(90)**
- **gap + dminutes(90)**
- **lap + dminutes(90)**

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

- **interval(nor, nor + minutes(90))**
- **interval(gap, gap + minutes(90))**
- **interval(lap, lap + minutes(90))**

**Intervals** are periods with the name of a time unit:

- **nor + minutes(90)**
- **gap + minutes(90)**
- **lap + 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(x = 1) x years.**
- **months(x = 1) x months.**
- **days(x = 1) x days.**
- **hours(x = 1) x hours.**
- **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.**

**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.

- **dd <- ddays(14)**
- **j <- ddays(14)**
- **n <- days(-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 `interval()` or `%-%`, e.g.

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

Not all days 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

```r
jan31 <- ymd(20180131)
```