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

d < date_decimal(2017.5)

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) 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 Sunday, Jan 17, 1999 3:34")

2. Apply the template to dates sf(ymd("2010-04-05"))

GET AND SET COMPONENTS

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

date(dt) Date component. date(dt)
year(x) Year. year(dt)
isoyear(x) The ISO 8601 year. isoiyear(x) dt)
quarter(x, with_year = FALSE) Quarter. quarter(dt)
semester(x, with_year = FALSE) Semester. semester(dt)
leap_year(x) Is it a leap year? leap_year(x)
leap_year(d) Is it a leap year? leap_year(x)

parse_date_time("9/1/01", "ymd")

Stamp Date-times

sf(stamp_date("2010-04-05"))

1. Derive a template, create a function sf (~ stamp("Created Sunday, Jan 17, 1999 3:34")

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()

OlsonNames()

OlsonNames()

OlsonNames()

OlsonNames()
Math with Date-times

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

**Periods**

Track changes in clock time, which ignore time line irregularities.

- A normal day: `nor <- ymd_hms("2018-01-01 01:30:00" ,tz="US/Eastern")`
- The start of daylight savings (spring forward): `gap <- ymd_hms("2018-03-11 01:30:00" ,tz="US/Eastern")`
- Leap years and leap seconds: `leap <- ymd("2019-03-01")`

**Durations**

Track the passage of physical time, which deviates from clock time when irregularities occur.

- The start of daylight savings (spring forward): `nor <- ymd_hms("2018-01-01 01:30:00" ,tz="US/Eastern")`
- A normal day: `nor + minutes(90)`
- Leap years and leap seconds: `leap + years(1)`

**Intervals**

Represent specific intervals of the timeline, bounded by start and end dates.

- `interval(nor, nor + minutes(90))`
- `interval(gap, gap + minutes(90))`
- `interval(leap, leap + years(1))`

**PERIODS**

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

**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 period with the name of a time unit, **pluralized**, e.g.:
  - `years(x = 1) x years`
  - `months(x = 1) x months`
  - `weeks(x = 1) x weeks`
  - `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`

- Make a period with the name of a time unit prefixed with a number, e.g.:
  - `years(2) x 2 years`
  - `months(3) + days(12) p
  - `3m 12d 0H 0M 0S`**

**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")`

- **Start** Date: `2017-01-01`
- **End** Date: `2017-12-31`

- `a %within% b` Does interval or date-time `a` fall within interval `b`? `now()`? `%within% i`
- `int_start(i)` Access/set the start date-time of an interval. Also `int_end()`, `as.interval()`, `is.interval()`, `int_length()`, `int_overlap()`, `int_standardize()`, `int_diff()`, `int_flip()`, `int_length()`, `int_shift()`