dateutil examples

dateutil.readthedocs.io/en/stable/examples.html

relativedelta examples

Let’s begin our trip:

```
>>> from datetime import *; from dateutil.relativedelta import *
>>> import calendar
```

Store some values:

```
>>> NOW = datetime.now()
>>> TODAY = date.today()
>>> NOW
datetime.datetime(2003, 9, 17, 20, 54, 47, 282310)
>>> TODAY
datetime.date(2003, 9, 17)
```

Next month

```
>>> NOW+relativedelta(months=+1)
datetime.datetime(2003, 10, 17, 20, 54, 47, 282310)
```

Next month, plus one week.

```
>>> NOW+relativedelta(months=+1, weeks=+1)
datetime.datetime(2003, 10, 24, 20, 54, 47, 282310)
```

Next month, plus one week, at 10am.

```
>>> TODAY+relativedelta(months=+1, weeks=+1, hour=10)
datetime.datetime(2003, 10, 24, 10, 0)
```

Here is another example using an absolute relativedelta. Notice the use of year and month (both singular) which causes the values to be *replaced* in the original datetime rather than performing an arithmetic operation on them.

```
>>> NOW+relativedelta(year=1, month=1)
datetime.datetime(1, 1, 17, 20, 54, 47, 282310)
```

Let’s try the other way around. Notice that the hour setting we get in the relativedelta is relative, since it’s a difference, and the weeks parameter has gone.

```
>>> relativedelta(datetime(2003, 10, 24, 10, 0), TODAY)
relativedelta(months=+1, days=+7, hours=+10)
```

One month before one year.

```
>>> NOW+relativedelta(years=+1, months=-1)
datetime.datetime(2004, 8, 17, 20, 54, 47, 282310)
```

How does it handle months with different numbers of days? Notice that adding one month will never cross the month boundary.

```
>>> date(2003,1,27)+relativedelta(months=+1)
datetime.date(2003, 2, 27)
>>> date(2003,1,31)+relativedelta(months=+1)
datetime.date(2003, 2, 28)
>>> date(2003,1,31)+relativedelta(months=+2)
datetime.date(2003, 3, 31)
```

The logic for years is the same, even on leap years.

```
>>> date(2000,2,28)+relativedelta(years=+1)
datetime.date(2001, 2, 28)
>>> date(2000,2,29)+relativedelta(years=+1)
datetime.date(2001, 2, 28)

>>> date(1999,2,28)+relativedelta(years=+1)
datetime.date(2000, 2, 28)
>>> date(1999,3,1)+relativedelta(years=+1)
datetime.date(2000, 3, 1)

>>> date(2001,2,28)+relativedelta(years=-1)
datetime.date(2000, 2, 28)
>>> date(2001,3,1)+relativedelta(years=-1)
datetime.date(2000, 3, 1)
```

Next friday

```
>>> TODAY+relativedelta(weekday=FR)
datetime.date(2003, 9, 19)

>>> TODAY+relativedelta(weekday=calendar.FRIDAY)
datetime.date(2003, 9, 19)
```

Last friday in this month.

```
>>> TODAY+relativedelta(day=31, weekday=FR(-1))
datetime.date(2003, 9, 26)
```

Next wednesday (it’s today!).

```
>>> TODAY+relativedelta(weekday=WE(+1))
datetime.date(2003, 9, 17)
```

Next wednesday, but not today.

```
>>> TODAY+relativedelta(days=+1, weekday=WE(+1))
datetime.date(2003, 9, 24)
```

Following ISO year week number notation find the first day of the 15th week of 1997.

```
>>> datetime(1997,1,1)+relativedelta(day=4, weekday=MO(-1), weeks=+14)
datetime.datetime(1997, 4, 7, 0, 0)
```

How long ago has the millennium changed?

```
>>> relativedelta(NOW, date(2001,1,1))
relativedelta(years=+2, months=+8, days=+16,
              hours=+20, minutes=+54, seconds=+47, microseconds=+282310)
```

How old is John?

```
>>> johnbirthday = datetime(1978, 4, 5, 12, 0)
>>> relativedelta(NOW, johnbirthday)
relativedelta(years=+25, months=+5, days=+12,
          hours=+8, minutes=+54, seconds=+47, microseconds=+282310)
```

It works with dates too.

```
>>> relativedelta(TODAY, johnbirthday)
relativedelta(years=+25, months=+5, days=+11, hours=+12)
```

Obtain today's date using the yearday:

```
>>> date(2003, 1, 1)+relativedelta(yearday=260)
datetime.date(2003, 9, 17)
```

We can use today's date, since yearday should be absolute in the given year:

```
>>> TODAY+relativedelta(yearday=260)
datetime.date(2003, 9, 17)
```

Last year it should be in the same day:

```
>>> date(2002, 1, 1)+relativedelta(yearday=260)
datetime.date(2002, 9, 17)
```

But not in a leap year:

```
>>> date(2000, 1, 1)+relativedelta(yearday=260)
datetime.date(2000, 9, 16)
```

We can use the non-leap year day to ignore this:

```
>>> date(2000, 1, 1)+relativedelta(nlyearday=260)
datetime.date(2000, 9, 17)
```

rrule examples

These examples were converted from the RFC.

Prepare the environment.

```
>>> from dateutil.rrule import *
>>> from dateutil.parser import *
>>> from datetime import *

>>> import pprint
>>> import sys
>>> sys.displayhook = pprint.pprint
```

Daily, for 10 occurrences.

```
>>> list(rrule(DAILY, count=10,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 5, 9, 0),
 datetime.datetime(1997, 9, 6, 9, 0),
 datetime.datetime(1997, 9, 7, 9, 0),
 datetime.datetime(1997, 9, 8, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 10, 9, 0),
 datetime.datetime(1997, 9, 11, 9, 0)]
```

Daily until December 24, 1997

```
>>> list(rrule(DAILY,
...            dtstart=parse("19970902T090000"),
...            until=parse("19971224T000000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 ...
 datetime.datetime(1997, 12, 21, 9, 0),
 datetime.datetime(1997, 12, 22, 9, 0),
 datetime.datetime(1997, 12, 23, 9, 0)]
```

Every other day, 5 occurrences.

```
>>> list(rrule(DAILY, interval=2, count=5,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 6, 9, 0),
 datetime.datetime(1997, 9, 8, 9, 0),
 datetime.datetime(1997, 9, 10, 9, 0)]
```

Every 10 days, 5 occurrences.

```
>>> list(rrule(DAILY, interval=10, count=5,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 12, 9, 0)]
```

Everyday in January, for 3 years.

```
>>> list(rrule(YEARLY, bymonth=1, byweekday=range(7),
...            dtstart=parse("19980101T090000"),
...            until=parse("20000131T090000")))
[datetime.datetime(1998, 1, 1, 9, 0),
 datetime.datetime(1998, 1, 2, 9, 0),
 ...
 datetime.datetime(1998, 1, 30, 9, 0),
 datetime.datetime(1998, 1, 31, 9, 0),
 datetime.datetime(1999, 1, 1, 9, 0),
 datetime.datetime(1999, 1, 2, 9, 0),
 ...
 datetime.datetime(1999, 1, 30, 9, 0),
 datetime.datetime(1999, 1, 31, 9, 0),
 datetime.datetime(2000, 1, 1, 9, 0),
 datetime.datetime(2000, 1, 2, 9, 0),
 ...
 datetime.datetime(2000, 1, 30, 9, 0),
 datetime.datetime(2000, 1, 31, 9, 0)]
```

Same thing, in another way.

```
>>> list(rrule(DAILY, bymonth=1,
...            dtstart=parse("19980101T090000"),
...            until=parse("20000131T090000")))
[datetime.datetime(1998, 1, 1, 9, 0),
 ...
 datetime.datetime(2000, 1, 31, 9, 0)]
```

Weekly for 10 occurrences.

```
>>> list(rrule(WEEKLY, count=10,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 7, 9, 0),
 datetime.datetime(1997, 10, 14, 9, 0),
 datetime.datetime(1997, 10, 21, 9, 0),
 datetime.datetime(1997, 10, 28, 9, 0),
 datetime.datetime(1997, 11, 4, 9, 0)]
```

Every other week, 6 occurrences.

```
>>> list(rrule(WEEKLY, interval=2, count=6,
...           dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 14, 9, 0),
 datetime.datetime(1997, 10, 28, 9, 0),
 datetime.datetime(1997, 11, 11, 9, 0)]
```

Weekly on Tuesday and Thursday for 5 weeks.

```
>>> list(rrule(WEEKLY, count=10, wkst=SU, byweekday=(TU,TH),
...           dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 11, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 18, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0),
 datetime.datetime(1997, 9, 25, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0)]
```

Every other week on Tuesday and Thursday, for 8 occurrences.

```
>>> list(rrule(WEEKLY, interval=2, count=8,
...           wkst=SU, byweekday=(TU,TH),
...           dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 18, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 14, 9, 0),
 datetime.datetime(1997, 10, 16, 9, 0)]
```

Monthly on the 1st Friday for ten occurrences.

```
>>> list(rrule(MONTHLY, count=10, byweekday=FR(1),
...           dtstart=parse("19970905T090000")))
[datetime.datetime(1997, 9, 5, 9, 0),
 datetime.datetime(1997, 10, 3, 9, 0),
 datetime.datetime(1997, 11, 7, 9, 0),
 datetime.datetime(1997, 12, 5, 9, 0),
 datetime.datetime(1998, 1, 2, 9, 0),
 datetime.datetime(1998, 2, 6, 9, 0),
 datetime.datetime(1998, 3, 6, 9, 0),
 datetime.datetime(1998, 4, 3, 9, 0),
 datetime.datetime(1998, 5, 1, 9, 0),
 datetime.datetime(1998, 6, 5, 9, 0)]
```

Every other month on the 1st and last Sunday of the month for 10 occurrences.

```
>>> list(rrule(MONTHLY, interval=2, count=10,
...            byweekday=(SU(1), SU(-1)),
...            dtstart=parse("19970907T090000")))
[datetime.datetime(1997, 9, 7, 9, 0),
 datetime.datetime(1997, 9, 28, 9, 0),
 datetime.datetime(1997, 11, 2, 9, 0),
 datetime.datetime(1997, 11, 30, 9, 0),
 datetime.datetime(1998, 1, 4, 9, 0),
 datetime.datetime(1998, 1, 25, 9, 0),
 datetime.datetime(1998, 3, 1, 9, 0),
 datetime.datetime(1998, 3, 29, 9, 0),
 datetime.datetime(1998, 5, 3, 9, 0),
 datetime.datetime(1998, 5, 31, 9, 0)]
```

Monthly on the second to last Monday of the month for 6 months.

```
>>> list(rrule(MONTHLY, count=6, byweekday=MO(-2),
...            dtstart=parse("19970922T090000")))
[datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 20, 9, 0),
 datetime.datetime(1997, 11, 17, 9, 0),
 datetime.datetime(1997, 12, 22, 9, 0),
 datetime.datetime(1998, 1, 19, 9, 0),
 datetime.datetime(1998, 2, 16, 9, 0)]
```

Monthly on the third to the last day of the month, for 6 months.

```
>>> list(rrule(MONTHLY, count=6, bymonthday=-3,
...            dtstart=parse("19970928T090000")))
[datetime.datetime(1997, 9, 28, 9, 0),
 datetime.datetime(1997, 10, 29, 9, 0),
 datetime.datetime(1997, 11, 28, 9, 0),
 datetime.datetime(1997, 12, 29, 9, 0),
 datetime.datetime(1998, 1, 29, 9, 0),
 datetime.datetime(1998, 2, 26, 9, 0)]
```

Monthly on the 2nd and 15th of the month for 5 occurrences.

```
>>> list(rrule(MONTHLY, count=5, bymonthday=(2,15),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 15, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 15, 9, 0),
 datetime.datetime(1997, 11, 2, 9, 0)]
```

Monthly on the first and last day of the month for 3 occurrences.

```
>>> list(rrule(MONTHLY, count=5, bymonthday=(-1,1,),
...           dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 1, 9, 0),
 datetime.datetime(1997, 10, 31, 9, 0),
 datetime.datetime(1997, 11, 1, 9, 0),
 datetime.datetime(1997, 11, 30, 9, 0)]
```

Every 18 months on the 10th thru 15th of the month for 10 occurrences.

```
>>> list(rrule(MONTHLY, interval=18, count=10,
...           bymonthday=range(10,16),
...           dtstart=parse("19970910T090000")))
[datetime.datetime(1997, 9, 10, 9, 0),
 datetime.datetime(1997, 9, 11, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 13, 9, 0),
 datetime.datetime(1997, 9, 14, 9, 0),
 datetime.datetime(1997, 9, 15, 9, 0),
 datetime.datetime(1999, 3, 10, 9, 0),
 datetime.datetime(1999, 3, 11, 9, 0),
 datetime.datetime(1999, 3, 12, 9, 0),
 datetime.datetime(1999, 3, 13, 9, 0)]
```

Every Tuesday, every other month, 6 occurrences.

```
>>> list(rrule(MONTHLY, interval=2, count=6, byweekday=TU,
...           dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 11, 4, 9, 0)]
```

Yearly in June and July for 10 occurrences.

```
>>> list(rrule(YEARLY, count=4, bymonth=(6,7),
...           dtstart=parse("19970610T090000")))
[datetime.datetime(1997, 6, 10, 9, 0),
 datetime.datetime(1997, 7, 10, 9, 0),
 datetime.datetime(1998, 6, 10, 9, 0),
 datetime.datetime(1998, 7, 10, 9, 0)]
```

Every 3rd year on the 1st, 100th and 200th day for 4 occurrences.

```
>>> list(rrule(YEARLY, count=4, interval=3, byyearday=(1,100,200),
...           dtstart=parse("19970101T090000")))
[datetime.datetime(1997, 1, 1, 9, 0),
 datetime.datetime(1997, 4, 10, 9, 0),
 datetime.datetime(1997, 7, 19, 9, 0),
 datetime.datetime(2000, 1, 1, 9, 0)]
```

Every 20th Monday of the year, 3 occurrences.

```
>>> list(rrule(YEARLY, count=3, byweekday=MO(20),
...            dtstart=parse("19970519T090000")))
[datetime.datetime(1997, 5, 19, 9, 0),
 datetime.datetime(1998, 5, 18, 9, 0),
 datetime.datetime(1999, 5, 17, 9, 0)]
```

Monday of week number 20 (where the default start of the week is Monday), 3 occurrences.

```
>>> list(rrule(YEARLY, count=3, byweekno=20, byweekday=MO,
...            dtstart=parse("19970512T090000")))
[datetime.datetime(1997, 5, 12, 9, 0),
 datetime.datetime(1998, 5, 11, 9, 0),
 datetime.datetime(1999, 5, 17, 9, 0)]
```

The week number 1 may be in the last year.

```
>>> list(rrule(WEEKLY, count=3, byweekno=1, byweekday=MO,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 12, 29, 9, 0),
 datetime.datetime(1999, 1, 4, 9, 0),
 datetime.datetime(2000, 1, 3, 9, 0)]
```

And the week numbers greater than 51 may be in the next year.

```
>>> list(rrule(WEEKLY, count=3, byweekno=52, byweekday=SU,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 12, 28, 9, 0),
 datetime.datetime(1998, 12, 27, 9, 0),
 datetime.datetime(2000, 1, 2, 9, 0)]
```

Only some years have week number 53:

```
>>> list(rrule(WEEKLY, count=3, byweekno=53, byweekday=MO,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1998, 12, 28, 9, 0),
 datetime.datetime(2004, 12, 27, 9, 0),
 datetime.datetime(2009, 12, 28, 9, 0)]
```

Every Friday the 13th, 4 occurrences.

```
>>> list(rrule(YEARLY, count=4, byweekday=FR, bymonthday=13,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1998, 2, 13, 9, 0),
 datetime.datetime(1998, 3, 13, 9, 0),
 datetime.datetime(1998, 11, 13, 9, 0),
 datetime.datetime(1999, 8, 13, 9, 0)]
```

Every four years, the first Tuesday after a Monday in November, 3 occurrences (U.S. Presidential Election day):

```
>>> list(rrule(YEARLY, interval=4, count=3, bymonth=11,
...             byweekday=TU, bymonthday=(2,3,4,5,6,7,8),
...             dtstart=parse("19961105T090000")))
[datetime.datetime(1996, 11, 5, 9, 0),
 datetime.datetime(2000, 11, 7, 9, 0),
 datetime.datetime(2004, 11, 2, 9, 0)]
```

The 3rd instance into the month of one of Tuesday, Wednesday or Thursday, for the next 3 months:

```
>>> list(rrule(MONTHLY, count=3, byweekday=(TU,WE,TH),
...             bysetpos=3, dtstart=parse("19970904T090000")))
[datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 10, 7, 9, 0),
 datetime.datetime(1997, 11, 6, 9, 0)]
```

The 2nd to last weekday of the month, 3 occurrences.

```
>>> list(rrule(MONTHLY, count=3, byweekday=(MO,TU,WE,TH,FR),
...             bysetpos=-2, dtstart=parse("19970929T090000")))
[datetime.datetime(1997, 9, 29, 9, 0),
 datetime.datetime(1997, 10, 30, 9, 0),
 datetime.datetime(1997, 11, 27, 9, 0)]
```

Every 3 hours from 9:00 AM to 5:00 PM on a specific day.

```
>>> list(rrule(HOURLY, interval=3,
...             dtstart=parse("19970902T090000"),
...             until=parse("19970902T170000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 12, 0),
 datetime.datetime(1997, 9, 2, 15, 0)]
```

Every 15 minutes for 6 occurrences.

```
>>> list(rrule(MINUTELY, interval=15, count=6,
...             dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 9, 15),
 datetime.datetime(1997, 9, 2, 9, 30),
 datetime.datetime(1997, 9, 2, 9, 45),
 datetime.datetime(1997, 9, 2, 10, 0),
 datetime.datetime(1997, 9, 2, 10, 15)]
```

Every hour and a half for 4 occurrences.

```
>>> list(rrule(MINUTELY, interval=90, count=4,
...             dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 10, 30),
 datetime.datetime(1997, 9, 2, 12, 0),
 datetime.datetime(1997, 9, 2, 13, 30)]
```

Every 20 minutes from 9:00 AM to 4:40 PM for two days.

```
>>> list(rrule(MINUTELY, interval=20, count=48,
...            byhour=range(9,17), byminute=(0,20,40),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 9, 20),
 ...
 datetime.datetime(1997, 9, 2, 16, 20),
 datetime.datetime(1997, 9, 2, 16, 40),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 20),
 ...
 datetime.datetime(1997, 9, 3, 16, 20),
 datetime.datetime(1997, 9, 3, 16, 40)]
```

An example where the days generated makes a difference because of *wkst*.

```
>>> list(rrule(WEEKLY, interval=2, count=4,
...            byweekday=(TU,SU), wkst=MO,
...            dtstart=parse("19970805T090000")))
[datetime.datetime(1997, 8, 5, 9, 0),
 datetime.datetime(1997, 8, 10, 9, 0),
 datetime.datetime(1997, 8, 19, 9, 0),
 datetime.datetime(1997, 8, 24, 9, 0)]

>>> list(rrule(WEEKLY, interval=2, count=4,
...            byweekday=(TU,SU), wkst=SU,
...            dtstart=parse("19970805T090000")))
[datetime.datetime(1997, 8, 5, 9, 0),
 datetime.datetime(1997, 8, 17, 9, 0),
 datetime.datetime(1997, 8, 19, 9, 0),
 datetime.datetime(1997, 8, 31, 9, 0)]
```

rruleset examples

Daily, for 7 days, jumping Saturday and Sunday occurrences.

```
>>> set = rruleset()
>>> set.rrule(rrule(DAILY, count=7,
...                 dtstart=parse("19970902T090000")))
>>> set.exrule(rrule(YEARLY, byweekday=(SA,SU),
...                  dtstart=parse("19970902T090000")))
>>> list(set)
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 5, 9, 0),
 datetime.datetime(1997, 9, 8, 9, 0)]
```

Weekly, for 4 weeks, plus one time on day 7, and not on day 16.

```
>>> set = rruleset()
>>> set.rrule(rrule(WEEKLY, count=4,
...           dtstart=parse("19970902T090000")))
>>> set.rdate(datetime.datetime(1997, 9, 7, 9, 0))
>>> set.exdate(datetime.datetime(1997, 9, 16, 9, 0))
>>> list(set)
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 7, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0)]
```

rrulestr() examples

Every 10 days, 5 occurrences.

```
>>> list(rrulestr("""
... DTSTART:19970902T090000
... RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
... """))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 12, 9, 0)]
```

Same thing, but passing only the *RRULE* value.

```
>>> list(rrulestr("FREQ=DAILY;INTERVAL=10;COUNT=5",
...               dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 12, 9, 0)]
```

Notice that when using a single rule, it returns an *rrule* instance, unless *forceset* was used.

```
>>> rrulestr("FREQ=DAILY;INTERVAL=10;COUNT=5")
<dateutil.rrule.rrule object at 0x...>

>>> rrulestr("""
... DTSTART:19970902T090000
... RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
... """)
<dateutil.rrule.rrule object at 0x...>

>>> rrulestr("FREQ=DAILY;INTERVAL=10;COUNT=5", forceset=True)
<dateutil.rrule.rruleset object at 0x...>
```

But when an *rruleset* is needed, it is automatically used.

```
>>> rrulestr("""
... DTSTART:19970902T090000
... RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
... RRULE:FREQ=DAILY;INTERVAL=5;COUNT=3
... """)
<dateutil.rrule.rruleset object at 0x...>
```

parse examples

The following code will prepare the environment:

```
>>> from dateutil.parser import *
>>> from dateutil.tz import *
>>> from datetime import *
>>> TZOFFSETS = {"BRST": -10800}
>>> BRSTTZ = tzoffset("BRST", -10800)
>>> DEFAULT = datetime(2003, 9, 25)
```

Some simple examples based on the *date* command, using the *ZOFFSET* dictionary to provide the BRST timezone offset.

```
>>> parse("Thu Sep 25 10:36:28 BRST 2003", tzinfos=TZOFFSETS)
datetime.datetime(2003, 9, 25, 10, 36, 28,
        tzinfo=tzoffset('BRST', -10800))

>>> parse("2003 10:36:28 BRST 25 Sep Thu", tzinfos=TZOFFSETS)
datetime.datetime(2003, 9, 25, 10, 36, 28,
        tzinfo=tzoffset('BRST', -10800))
```

Notice that since BRST is my local timezone, parsing it without further timezone settings will yield a *tzlocal* timezone.

```
>>> parse("Thu Sep 25 10:36:28 BRST 2003")
datetime.datetime(2003, 9, 25, 10, 36, 28, tzinfo=tzlocal())
```

We can also ask to ignore the timezone explicitly:

```
>>> parse("Thu Sep 25 10:36:28 BRST 2003", ignoretz=True)
datetime.datetime(2003, 9, 25, 10, 36, 28)
```

That’s the same as processing a string without timezone:

```
>>> parse("Thu Sep 25 10:36:28 2003")
datetime.datetime(2003, 9, 25, 10, 36, 28)
```

Without the year, but passing our *DEFAULT* datetime to return the same year, no mattering what year we currently are in:

```
>>> parse("Thu Sep 25 10:36:28", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36, 28)
```

Strip it further:

```
>>> parse("Thu Sep 10:36:28", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36, 28)

>>> parse("Thu 10:36:28", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36, 28)

>>> parse("Thu 10:36", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36)

>>> parse("10:36", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36)
```

Strip in a different way:

```
>>> parse("Thu Sep 25 2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("Sep 25 2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("Sep 2003", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("Sep", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("2003", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)
```

Another format, based on *date -R* (RFC822):

```
 >>> parse("Thu, 25 Sep 2003 10:49:41 -0300")
 datetime.datetime(2003, 9, 25, 10, 49, 41,
          tzinfo=tzoffset(None, -10800))
```

ISO format:

```
 >>> parse("2003-09-25T10:49:41.5-03:00")
 datetime.datetime(2003, 9, 25, 10, 49, 41, 500000,
          tzinfo=tzoffset(None, -10800))
```

Some variations:

```
>>> parse("2003-09-25T10:49:41")
datetime.datetime(2003, 9, 25, 10, 49, 41)

>>> parse("2003-09-25T10:49")
datetime.datetime(2003, 9, 25, 10, 49)

>>> parse("2003-09-25T10")
datetime.datetime(2003, 9, 25, 10, 0)

>>> parse("2003-09-25")
datetime.datetime(2003, 9, 25, 0, 0)
```

ISO format, without separators:

```
>>> parse("20030925T104941.5-0300")
datetime.datetime(2003, 9, 25, 10, 49, 41, 500000,
                  tzinfo=tzoffset(None, -10800))

>>> parse("20030925T104941-0300")
datetime.datetime(2003, 9, 25, 10, 49, 41,
          tzinfo=tzoffset(None, -10800))

>>> parse("20030925T104941")
datetime.datetime(2003, 9, 25, 10, 49, 41)

>>> parse("20030925T1049")
datetime.datetime(2003, 9, 25, 10, 49)

>>> parse("20030925T10")
datetime.datetime(2003, 9, 25, 10, 0)

>>> parse("20030925")
datetime.datetime(2003, 9, 25, 0, 0)
```

Everything together.

```
>>> parse("199709020900")
datetime.datetime(1997, 9, 2, 9, 0)
>>> parse("19970902090059")
datetime.datetime(1997, 9, 2, 9, 0, 59)
```

Different date orderings:

```
>>> parse("2003-09-25")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("2003-Sep-25")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("25-Sep-2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("Sep-25-2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("09-25-2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("25-09-2003")
datetime.datetime(2003, 9, 25, 0, 0)
```

Check some ambiguous dates:

```
>>> parse("10-09-2003")
datetime.datetime(2003, 10, 9, 0, 0)

>>> parse("10-09-2003", dayfirst=True)
datetime.datetime(2003, 9, 10, 0, 0)

>>> parse("10-09-03")
datetime.datetime(2003, 10, 9, 0, 0)

>>> parse("10-09-03", yearfirst=True)
datetime.datetime(2010, 9, 3, 0, 0)
```

Other date separators are allowed:

```
>>> parse("2003.Sep.25")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("2003/09/25")
datetime.datetime(2003, 9, 25, 0, 0)
```

Even with spaces:

```
>>> parse("2003 Sep 25")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("2003 09 25")
datetime.datetime(2003, 9, 25, 0, 0)
```

Hours with letters work:

```
>>> parse("10h36m28.5s", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36, 28, 500000)

>>> parse("01s02h03m", default=DEFAULT)
datetime.datetime(2003, 9, 25, 2, 3, 1)

>>> parse("01h02m03", default=DEFAULT)
datetime.datetime(2003, 9, 25, 1, 2, 3)

>>> parse("01h02", default=DEFAULT)
datetime.datetime(2003, 9, 25, 1, 2)

>>> parse("01h02s", default=DEFAULT)
datetime.datetime(2003, 9, 25, 1, 0, 2)
```

With AM/PM:

```
>>> parse("10h am", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 0)

>>> parse("10pm", default=DEFAULT)
datetime.datetime(2003, 9, 25, 22, 0)

>>> parse("12:00am", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("12pm", default=DEFAULT)
datetime.datetime(2003, 9, 25, 12, 0)
```

Some special treating for ‘’pertain’’ relations:

```
>>> parse("Sep 03", default=DEFAULT)
datetime.datetime(2003, 9, 3, 0, 0)

>>> parse("Sep of 03", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)
```

Fuzzy parsing:

```
 >>> s = "Today is 25 of September of 2003, exactly " \
 ...     "at 10:49:41 with timezone -03:00."
 >>> parse(s, fuzzy=True)
 datetime.datetime(2003, 9, 25, 10, 49, 41,
           tzinfo=tzoffset(None, -10800))
```

Other random formats:

```
>>> parse("Wed, July 10, '96")
datetime.datetime(1996, 7, 10, 0, 0)

>>> parse("1996.07.10 AD at 15:08:56 PDT", ignoretz=True)
datetime.datetime(1996, 7, 10, 15, 8, 56)

>>> parse("Tuesday, April 12, 1952 AD 3:30:42pm PST", ignoretz=True)
datetime.datetime(1952, 4, 12, 15, 30, 42)

>>> parse("November 5, 1994, 8:15:30 am EST", ignoretz=True)
datetime.datetime(1994, 11, 5, 8, 15, 30)

>>> parse("3rd of May 2001")
datetime.datetime(2001, 5, 3, 0, 0)

>>> parse("5:50 A.M. on June 13, 1990")
datetime.datetime(1990, 6, 13, 5, 50)
```

Override parserinfo with a custom parserinfo

```
>>> from dateutil.parser import parse, parserinfo
>>> class CustomParserInfo(parserinfo):
...     # e.g. edit a property of parserinfo to allow a custom 12 hour format
...     AMPM = [("am", "a", "xm"), ("pm", "p")]
>>> parse('2018-06-08 12:06:58 XM', parserinfo=CustomParserInfo())
datetime.datetime(2018, 6, 8, 0, 6, 58)
```

tzutc examples

```
>>> from datetime import *
>>> from dateutil import tz

>>> datetime.now()
datetime.datetime(2003, 9, 27, 9, 40, 1, 521290)

>>> datetime.now(tz.UTC)
datetime.datetime(2003, 9, 27, 12, 40, 12, 156379, tzinfo=tzutc())

>>> datetime.now(tz.UTC).tzname()
'UTC'
```

tzoffset examples

```
>>> from datetime import *
>>> from dateutil.tz import *

>>> datetime.now(tzoffset("BRST", -10800))
datetime.datetime(2003, 9, 27, 9, 52, 43, 624904,
         tzinfo=tzinfo=tzoffset('BRST', -10800))

>>> datetime.now(tzoffset("BRST", -10800)).tzname()
'BRST'

>>> datetime.now(tzoffset("BRST", -10800)).astimezone(UTC)
datetime.datetime(2003, 9, 27, 12, 53, 11, 446419,
         tzinfo=tzutc())
```

tzlocal examples

```
>>> from datetime import *
>>> from dateutil.tz import *

>>> datetime.now(tzlocal())
datetime.datetime(2003, 9, 27, 10, 1, 43, 673605,
         tzinfo=tzlocal())

>>> datetime.now(tzlocal()).tzname()
'BRST'

>>> datetime.now(tzlocal()).astimezone(tzoffset(None, 0))
datetime.datetime(2003, 9, 27, 13, 3, 0, 11493,
         tzinfo=tzoffset(None, 0))
```

tzstr examples

Here are examples of the recognized formats:

> - *EST5EDT*
> - *EST5EDT4,M4.1.0/02:00:00,M10-5-0/02:00*
> - *EST5EDT4,95/02:00:00,298/02:00*
> - *EST5EDT4,J96/02:00:00,J299/02:00*

Notice that if daylight information is not present, but a daylight abbreviation was provided, *tzstr* will follow the convention of using the first sunday of April to start daylight saving, and the last sunday of October to end it. If start or end time is not present, 2AM will be used, and if the daylight offset is not present, the standard offset plus one hour will be used. This convention is the same as used in the GNU libc.

This also means that some of the above examples are exactly equivalent, and all of these examples are equivalent in the year of 2003.

Here is the example mentioned in the

[https://docs.python.org/3/library/time.html time module documentation].

```
>>> os.environ['TZ'] = 'EST+05EDT,M4.1.0,M10.5.0'
>>> time.tzset()
>>> time.strftime('%X %x %Z')
'02:07:36 05/08/03 EDT'
>>> os.environ['TZ'] = 'AEST-10AEDT-11,M10.5.0,M3.5.0'
>>> time.tzset()
>>> time.strftime('%X %x %Z')
'16:08:12 05/08/03 AEST'
```

And here is an example showing the same information using *tzstr*, without touching system settings.

```
>>> tz1 = tzstr('EST+05EDT,M4.1.0,M10.5.0')
>>> tz2 = tzstr('AEST-10AEDT-11,M10.5.0,M3.5.0')
>>> dt = datetime(2003, 5, 8, 2, 7, 36, tzinfo=tz1)
>>> dt.strftime('%X %x %Z')
'02:07:36 05/08/03 EDT'
>>> dt.astimezone(tz2).strftime('%X %x %Z')
'16:07:36 05/08/03 AEST'
```

Are these really equivalent?

```
>>> tzstr('EST5EDT') == tzstr('EST5EDT,M4.1.0,M10.5.0')
True
```

Check the daylight limit.

```
>>> tz = tzstr('EST+05EDT,M4.1.0,M10.5.0')
>>> datetime(2003, 4, 6, 1, 59, tzinfo=tz).tzname()
'EST'
>>> datetime(2003, 4, 6, 2, 00, tzinfo=tz).tzname()
'EDT'
>>> datetime(2003, 10, 26, 0, 59, tzinfo=tz).tzname()
'EDT'
>>> datetime(2003, 10, 26, 2, 00, tzinfo=tz).tzname()
'EST'
```

tzrange examples

```
>>> tzstr('EST5EDT') == tzrange("EST", -18000, "EDT")
True

>>> from dateutil.relativedelta import *
>>> range1 = tzrange("EST", -18000, "EDT")
>>> range2 = tzrange("EST", -18000, "EDT", -14400,
...                  relativedelta(hours=+2, month=4, day=1,
...                                weekday=SU(+1)),
...                  relativedelta(hours=+1, month=10, day=31,
...                                weekday=SU(-1)))
>>> tzstr('EST5EDT') == range1 == range2
True
```

Notice a minor detail in the last example: while the DST should end at 2AM, the delta will catch 1AM. That’s because the daylight saving time should end at 2AM standard time (the difference between STD and DST is 1h in the given example) instead of the DST time. That’s how the *tzinfo* subtypes should deal with the extra hour that happens when going back to the standard time. Check

[https://docs.python.org/3/library/datetime.html#datetime.tzinfo tzinfo documentation]

for more information.

tzfile examples

```
>>> tz = tzfile("/etc/localtime")
>>> datetime.now(tz)
datetime.datetime(2003, 9, 27, 12, 3, 48, 392138,
          tzinfo=tzfile('/etc/localtime'))

>>> datetime.now(tz).astimezone(UTC)
datetime.datetime(2003, 9, 27, 15, 3, 53, 70863,
          tzinfo=tzutc())

>>> datetime.now(tz).tzname()
'BRST'
>>> datetime(2003, 1, 1, tzinfo=tz).tzname()
'BRDT'
```

Check the daylight limit.

```
>>> tz = tzfile('/usr/share/zoneinfo/EST5EDT')
>>> datetime(2003, 4, 6, 1, 59, tzinfo=tz).tzname()
'EST'
>>> datetime(2003, 4, 6, 2, 00, tzinfo=tz).tzname()
'EDT'
>>> datetime(2003, 10, 26, 0, 59, tzinfo=tz).tzname()
'EDT'
>>> datetime(2003, 10, 26, 1, 00, tzinfo=tz).tzname()
'EST'
```

tzical examples

Here is a sample file extracted from the RFC. This file defines the *EST5EDT* timezone, and will be used in the following example.

```
BEGIN:VTIMEZONE
TZID:US-Eastern
LAST-MODIFIED:19870101T000000Z
TZURL:http://zones.stds_r_us.net/tz/US-Eastern
BEGIN:STANDARD
DTSTART:19671029T020000
RRULE:FREQ=YEARLY;BYDAY=-1SU;BYMONTH=10
TZOFFSETFROM:-0400
TZOFFSETTO:-0500
TZNAME:EST
END:STANDARD
BEGIN:DAYLIGHT
DTSTART:19870405T020000
RRULE:FREQ=YEARLY;BYDAY=1SU;BYMONTH=4
TZOFFSETFROM:-0500
TZOFFSETTO:-0400
TZNAME:EDT
END:DAYLIGHT
END:VTIMEZONE
```

And here is an example exploring a *tzical* type:

```
>>> from dateutil.tz import *; from datetime import *

>>> tz = tzical('samples/EST5EDT.ics')
>>> tz.keys()
['US-Eastern']

>>> est = tz.get('US-Eastern')
>>> est
<tzicalvtz 'US-Eastern'>

>>> datetime.now(est)
datetime.datetime(2003, 10, 6, 19, 44, 18, 667987,
        tzinfo=<tzicalvtz 'US-Eastern'>)

>>> est == tz.get()
True
```

Let’s check the daylight ranges, as usual:

```
>>> datetime(2003, 4, 6, 1, 59, tzinfo=est).tzname()
'EST'
>>> datetime(2003, 4, 6, 2, 00, tzinfo=est).tzname()
'EDT'

>>> datetime(2003, 10, 26, 0, 59, tzinfo=est).tzname()
'EDT'
>>> datetime(2003, 10, 26, 1, 00, tzinfo=est).tzname()
'EST'
```

tzwin examples

```
>>> tz = tzwin("E. South America Standard Time")
```

tzwinlocal examples

```
>>> tz = tzwinlocal()
```

# vim:ts=4:sw=4:et