# datetime — Date and Time Value Manipulation

**Purpose:** The datetime module includes functions and classes for doing date and time parsing, formatting, and arithmetic.

`datetime` contains functions and classes for working with dates and times, separately and together.

## Times

Time values are represented with the `time` class. A `time` instance has attributes for `hour`, `minute`, `second`, and `microsecond` and can also include time zone information.

```
# datetime_time.py

import datetime

t = datetime.time(1, 2, 3)
print(t)
print('hour       :', t.hour)
print('minute     :', t.minute)
print('second     :', t.second)
print('microsecond:', t.microsecond)
print('tzinfo     :', t.tzinfo)
```

The arguments to initialize a `time` instance are optional, but the default of `0` is unlikely to be correct.

```
$ python3 datetime_time.py

01:02:03
hour       : 1
minute     : 2
second     : 3
microsecond: 0
tzinfo     : None
```

A `time` instance only holds values of time, and not a date associated with the time.

```
# datetime_time_minmax.py

import datetime

print('Earliest  :', datetime.time.min)
print('Latest    :', datetime.time.max)
print('Resolution:', datetime.time.resolution)
```

The `min` and `max` class attributes reflect the valid range of times in a single day.

```
$ python3 datetime_time_minmax.py

Earliest  : 00:00:00
Latest    : 23:59:59.999999
Resolution: 0:00:00.000001
```

The resolution for `time` is limited to whole microseconds.

```
# datetime_time_resolution.py

import datetime

for m in [1, 0, 0.1, 0.6]:
    try:
```

```
        print('{:02.1f} :'.format(m),
              datetime.time(0, 0, 0, microsecond=m))
    except TypeError as err:
        print('ERROR:', err)
```

Floating point values for microseconds cause a `TypeError`.

```
$ python3 datetime_time_resolution.py

1.0 : 00:00:00.000001
0.0 : 00:00:00
ERROR: integer argument expected, got float
ERROR: integer argument expected, got float
```

## Dates

Calendar date values are represented with the `date` class. Instances have attributes for `year`, `month`, and `day`. It is easy to create a date representing the current date using the `today()` class method.

```
# datetime_date.py

import datetime

today = datetime.date.today()
print(today)
print('ctime  :', today.ctime())
tt = today.timetuple()
print('tuple  : tm_year  =', tt.tm_year)
print('         tm_mon   =', tt.tm_mon)
print('         tm_mday  =', tt.tm_mday)
print('         tm_hour  =', tt.tm_hour)
print('         tm_min   =', tt.tm_min)
print('         tm_sec   =', tt.tm_sec)
print('         tm_wday  =', tt.tm_wday)
print('         tm_yday  =', tt.tm_yday)
print('         tm_isdst =', tt.tm_isdst)
print('ordinal:', today.toordinal())
print('Year   :', today.year)
print('Mon    :', today.month)
print('Day    :', today.day)
```

This example prints the current date in several formats:

```
$ python3 datetime_date.py

2018-12-09
ctime  : Sun Dec  9 00:00:00 2018
tuple  : tm_year  = 2018
         tm_mon   = 12
         tm_mday  = 9
         tm_hour  = 0
         tm_min   = 0
         tm_sec   = 0
         tm_wday  = 6
         tm_yday  = 343
         tm_isdst = -1
ordinal: 737037
Year   : 2018
Mon    : 12
Day    : 9
```

There are also class methods for creating instances from POSIX timestamps or integers representing date values from the Gregorian calendar, where January 1 of the year 1 is 1 and each subsequent day increments the value by 1.

```
# datetime_date_fromordinal.py

import datetime
import time
```

```
o = 733114
print('o               :', o)
print('fromordinal(o)  :', datetime.date.fromordinal(o))

t = time.time()
print('t               :', t)
print('fromtimestamp(t):', datetime.date.fromtimestamp(t))
```

This example illustrates the different value types used by `fromordinal()` and `fromtimestamp()`.

```
$ python3 datetime_date_fromordinal.py

o               : 733114
fromordinal(o)  : 2008-03-13
t               : 1544370390.0430489
fromtimestamp(t): 2018-12-09
```

As with `time`, the range of date values supported can be determined using the `min` and `max` attributes.

```
# datetime_date_minmax.py

import datetime

print('Earliest  :', datetime.date.min)
print('Latest    :', datetime.date.max)
print('Resolution:', datetime.date.resolution)
```

The resolution for dates is whole days.

```
$ python3 datetime_date_minmax.py

Earliest  : 0001-01-01
Latest    : 9999-12-31
Resolution: 1 day, 0:00:00
```

Another way to create new `date` instances uses the `replace()` method of an existing `date`.

```
# datetime_date_replace.py

import datetime

d1 = datetime.date(2008, 3, 29)
print('d1:', d1.ctime())

d2 = d1.replace(year=2009)
print('d2:', d2.ctime())
```

This example changes the year, leaving the day and month unmodified.

```
$ python3 datetime_date_replace.py

d1: Sat Mar 29 00:00:00 2008
d2: Sun Mar 29 00:00:00 2009
```

## timedeltas

Future and past dates can be calculated using basic arithmetic on two `datetime` objects, or by combining a `datetime` with a `timedelta`. Subtracting dates produces a `timedelta`, and a `timedelta` can be added or subtracted from a date to produce another date. The internal values for a `timedelta` are stored in days, seconds, and microseconds.

```
# datetime_timedelta.py

import datetime

print('microseconds:', datetime.timedelta(microseconds=1))
print('milliseconds:', datetime.timedelta(milliseconds=1))
```

```
print('seconds     :', datetime.timedelta(seconds=1))
print('minutes     :', datetime.timedelta(minutes=1))
print('hours       :', datetime.timedelta(hours=1))
print('days        :', datetime.timedelta(days=1))
print('weeks       :', datetime.timedelta(weeks=1))
```

Intermediate level values passed to the constructor are converted into days, seconds, and microseconds.

```
$ python3 datetime_timedelta.py

microseconds: 0:00:00.000001
milliseconds: 0:00:00.001000
seconds     : 0:00:01
minutes     : 0:01:00
hours       : 1:00:00
days        : 1 day, 0:00:00
weeks       : 7 days, 0:00:00
```

The full duration of a `timedelta` can be retrieved as a number of seconds using `total_seconds()`.

```
# datetime_timedelta_total_seconds.py

import datetime

for delta in [datetime.timedelta(microseconds=1),
              datetime.timedelta(milliseconds=1),
              datetime.timedelta(seconds=1),
              datetime.timedelta(minutes=1),
              datetime.timedelta(hours=1),
              datetime.timedelta(days=1),
              datetime.timedelta(weeks=1),
              ]:
    print('{:15} = {:8} seconds'.format(
        str(delta), delta.total_seconds())
    )
```

The return value is a floating point number, to accommodate sub-second durations.

```
$ python3 datetime_timedelta_total_seconds.py

0:00:00.000001  =    1e-06 seconds
0:00:00.001000  =    0.001 seconds
0:00:01         =      1.0 seconds
0:01:00         =     60.0 seconds
1:00:00         =   3600.0 seconds
1 day, 0:00:00  =  86400.0 seconds
7 days, 0:00:00 = 604800.0 seconds
```

## Date Arithmetic

Date math uses the standard arithmetic operators.

```
# datetime_date_math.py

import datetime

today = datetime.date.today()
print('Today    :', today)

one_day = datetime.timedelta(days=1)
print('One day  :', one_day)

yesterday = today - one_day
print('Yesterday:', yesterday)

tomorrow = today + one_day
print('Tomorrow :', tomorrow)

print()
```

```
print()
print('tomorrow - yesterday:', tomorrow - yesterday)
print('yesterday - tomorrow:', yesterday - tomorrow)
```

This example with date objects illustrates using `timedelta` objects to compute new dates, and subtracting date instances to produce timedeltas (including a negative delta value).

```
$ python3 datetime_date_math.py

Today    : 2018-12-09
One day  : 1 day, 0:00:00
Yesterday: 2018-12-08
Tomorrow : 2018-12-10

tomorrow - yesterday: 2 days, 0:00:00
yesterday - tomorrow: -2 days, 0:00:00
```

A `timedelta` object also supports arithmetic with integers, floats, and other `timedelta` instances.

```
# datetime_timedelta_math.py

import datetime

one_day = datetime.timedelta(days=1)
print('1 day    :', one_day)
print('5 days   :', one_day * 5)
print('1.5 days :', one_day * 1.5)
print('1/4 day  :', one_day / 4)

# assume an hour for lunch
work_day = datetime.timedelta(hours=7)
meeting_length = datetime.timedelta(hours=1)
print('meetings per day :', work_day / meeting_length)
```

In this example, several multiples of a single day are computed, with the resulting `timedelta` holding the appropriate number of days or hours. The final example demonstrates how to compute values by combining two `timedelta` objects. In this case, the result is a floating point number.

```
$ python3 datetime_timedelta_math.py

1 day    : 1 day, 0:00:00
5 days   : 5 days, 0:00:00
1.5 days : 1 day, 12:00:00
1/4 day  : 6:00:00
meetings per day : 7.0
```

## Comparing Values

Both date and time values can be compared using the standard comparison operators to determine which is earlier or later.

```
# datetime_comparing.py

import datetime
import time

print('Times:')
t1 = datetime.time(12, 55, 0)
print('  t1:', t1)
t2 = datetime.time(13, 5, 0)
print('  t2:', t2)
print('  t1 < t2:', t1 < t2)

print()
print('Dates:')
d1 = datetime.date.today()
print('  d1:', d1)
d2 = datetime.date.today() + datetime.timedelta(days=1)
print('  d2:', d2)
print('  d1 > d2:', d1 > d2)
```

```
print('   d1 > d2:', d1 > d2)
```

All comparison operators are supported.

```
$ python3 datetime_comparing.py

Times:
  t1: 12:55:00
  t2: 13:05:00
  t1 < t2: True

Dates:
  d1: 2018-12-09
  d2: 2018-12-10
  d1 > d2: False
```

## Combining Dates and Times

Use the `datetime` class to hold values consisting of both date and time components. As with `date`, there are several convenient class methods to make creating `datetime` instances from other common values.

```
# datetime_datetime.py

import datetime

print('Now    :', datetime.datetime.now())
print('Today  :', datetime.datetime.today())
print('UTC Now:', datetime.datetime.utcnow())
print()

FIELDS = [
    'year', 'month', 'day',
    'hour', 'minute', 'second',
    'microsecond',
]

d = datetime.datetime.now()
for attr in FIELDS:
    print('{:15}: {}'.format(attr, getattr(d, attr)))
```

As might be expected, the `datetime` instance has all of the attributes of both a `date` and a `time` object.

```
$ python3 datetime_datetime.py

Now    : 2018-12-09 10:46:30.494767
Today  : 2018-12-09 10:46:30.494806
UTC Now: 2018-12-09 15:46:30.494812

year           : 2018
month          : 12
day            : 9
hour           : 10
minute         : 46
second         : 30
microsecond    : 495051
```

Just as with `date`, `datetime` provides convenient class methods for creating new instances. It also includes `fromordinal()` and `fromtimestamp()`.

```
# datetime_datetime_combine.py

import datetime

t = datetime.time(1, 2, 3)
print('t :', t)

d = datetime.date.today()
print('d :', d)
```

```
dt = datetime.datetime.combine(d, t)
print('dt:', dt)
```

`combine()` creates `datetime` instances from one `date` and one `time` instance.

```
$ python3 datetime_datetime_combine.py

t : 01:02:03
d : 2018-12-09
dt: 2018-12-09 01:02:03
```

## Formatting and Parsing

The default string representation of a datetime object uses the ISO-8601 format (`YYYY-MM-DDTHH:MM:SS.mmmmmm`). Alternate formats can be generated using `strftime()`.

```
# datetime_datetime_strptime.py

import datetime

format = "%a %b %d %H:%M:%S %Y"

today = datetime.datetime.today()
print('ISO     :', today)

s = today.strftime(format)
print('strftime:', s)

d = datetime.datetime.strptime(s, format)
print('strptime:', d.strftime(format))
```

Use `datetime.strptime()` to convert formatted strings to `datetime` instances.

```
$ python3 datetime_datetime_strptime.py

ISO     : 2018-12-09 10:46:30.598115
strftime: Sun Dec 09 10:46:30 2018
strptime: Sun Dec 09 10:46:30 2018
```

The same formatting codes can be used with Python's string formatting mini-language by placing them after the `:` in the field specification of the format string.

```
# datetime_format.py

import datetime

today = datetime.datetime.today()
print('ISO     :', today)
print('format(): {:%a %b %d %H:%M:%S %Y}'.format(today))
```

Each datetime format code must still be prefixed with `%`, and subsequent colons are treated as literal characters to include in the output.

```
$ python3 datetime_format.py

ISO     : 2018-12-09 10:46:30.659964
format(): Sun Dec 09 10:46:30 2018
```

The following table demonstrates all of the formatting codes for 5:00 PM January 13, 2016 in the US/Eastern time zone.

strptime/strftime format codes

| Symbol | Meaning | Example |
|---|---|---|
| %a | Abbreviated weekday name | 'Wed' |
| %A | Full weekday name | 'Wednesday' |
| %w | Weekday number – 0 (Sunday) through 6 (Saturday) | '3' |

| | | |
|---|---|---|
| %d | Day of the month (zero padded) | '13' |
| %b | Abbreviated month name | 'Jan' |
| %B | Full month name | 'January' |
| %m | Month of the year | '01' |
| %y | Year without century | '16' |
| %Y | Year with century | '2016' |
| %H | Hour from 24-hour clock | '17' |
| %I | Hour from 12-hour clock | '05' |
| %p | AM/PM | 'PM' |
| %M | Minutes | '00' |
| %S | Seconds | '00' |
| %f | Microseconds | '000000' |
| %z | UTC offset for time zone-aware objects | '-0500' |
| %Z | Time Zone name | 'EST' |
| %j | Day of the year | '013' |
| %W | Week of the year | '02' |
| %c | Date and time representation for the current locale | 'Wed Jan 13 17:00:00 2016' |
| %x | Date representation for the current locale | '01/13/16' |
| %X | Time representation for the current locale | '17:00:00' |
| %% | A literal % character | '%' |

## Time Zones

Within `datetime`, time zones are represented by subclasses of `tzinfo`. Since `tzinfo` is an abstract base class, applications need to define a subclass and provide appropriate implementations for a few methods to make it useful.

`datetime` does include a somewhat naive implementation in the class `timezone` that uses a fixed offset from UTC, and does not support different offset values on different days of the year, such as where daylight savings time applies, or where the offset from UTC has changed over time.

```
# datetime_timezone.py

import datetime

min6 = datetime.timezone(datetime.timedelta(hours=-6))
plus6 = datetime.timezone(datetime.timedelta(hours=6))
d = datetime.datetime.now(min6)

print(min6, ':', d)
print(datetime.timezone.utc, ':',
      d.astimezone(datetime.timezone.utc))
print(plus6, ':', d.astimezone(plus6))

# convert to the current system timezone
d_system = d.astimezone()
print(d_system.tzinfo, '      :', d_system)
```

To convert a datetime value from one time zone to another, use `astimezone()`. In the example above, two separate time zones 6 hours on either side of UTC are shown, and the `utc` instance from `datetime.timezone` is also used for reference. The final output line shows the value in the system timezone, acquired by calling `astimezone()` with no argument.

```
$ python3 datetime_timezone.py

UTC-06:00 : 2018-12-09 09:46:30.709455-06:00
UTC : 2018-12-09 15:46:30.709455+00:00
UTC+06:00 : 2018-12-09 21:46:30.709455+06:00
EST       : 2018-12-09 10:46:30.709455-05:00
```

**Note**

The third party module pytz is a better implementation for time zones. It supports named time zones, and the offset database is kept up to date as changes are made by political bodies around the world.

## See also

- Standard library documentation for datetime
- Python 2 to 3 porting notes for datetime
- `calendar` – The `calendar` module.
- `time` – The `time` module.
- dateutil – `dateutil` from Labix extends the `datetime` module with additional features.
- pytz – World Time Zone database and classes for making `datetime` objects time zone-aware.
- WikiPedia: Proleptic Gregorian calendar – A description of the Gregorian calendar system.
- ISO 8601 – The standard for numeric representation of Dates and Time

time — Clock Time

calendar — Work with Dates

**Quick Links**

Times
Dates
timedeltas
Date Arithmetic
Comparing Values
Combining Dates and Times
Formatting and Parsing
Time Zones

*This page was last updated 2018-12-09.*

**Navigation**

time — Clock Time
calendar — Work with Dates



*The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.*

*Looking for examples for Python 2?*

**This Site**

Module Index
Index

**Other Writing**

- Blog
- The Python Standard Library By Example