017 Python: Date and Time

Date and Time

The datetime module supplies classes for manipulating dates and times in both simple and complex ways.

Basic datetime objects usage:

The datetime module contains three primary types of objects - date, time, and datetime.

Date:

```
import datetime
today = datetime.date.today()
new_year = datetime.date(2019, 1, 1)
print(new_year)
```

Output:

```
2019-01-01
```

Time:

```
import datetime
#Time object
noon = datetime.time(12, 0, 0)
print(noon)
```

Output:

```
12:00:00
```

Date Time:

```
import datetime
# Current datetime
now = datetime.datetime.now()
print(now)
```

Output:

```
2019-11-01 06:16:18.526734
```

Date Time:

```
import datetime
# Datetime object
millenium_turn = datetime.datetime(2019, 1, 1, 0, 0, 0)
print(millenium_turn)
```

Output:

```
2019-01-01 00:00:00
```

Iterate over dates:

Print from a start date to some end date.

```
import datetime

# The size of each step in days
day_delta = datetime.timedelta(days=1)

start_date = datetime.date.today()
end_date = start_date + 7*day_delta

for i in range((end_date - start_date).days):
    print(start_date + i*day_delta)
```

Output:

```
2019-11-01
2019-11-02
2019-11-03
2019-11-04
2019-11-05
2019-11-06
2019-11-07
```

Computing time differences:

The timedelta module is used to compute differences between times:

```
from datetime import datetime, timedelta
now = datetime.now()
then = datetime(2019, 5, 23)
print(then)
```

Output:

```
2019-05-23 00:00:00
```

Specifying time is optional when creating a new datetime object

```
from datetime import datetime, timedelta
now = datetime.now()
then = datetime(2019, 5, 23)
delta = now-then
print(delta)
```

Output:

```
162 days, 9:10:42.599772
```

delta is of type timedelta:

```
from datetime import datetime, timedelta
now = datetime.now()
then = datetime(2019, 5, 23)
delta = now-then
print(delta.days)
# 60
print(delta.seconds)
# 40826
```

Output:

```
162
33296
```

To get n day's after and n day's before date we could use:

n day's after date:

```
from datetime import date, timedelta

current_date = date.today().isoformat()
days_after = (date.today()+timedelta(days=30)).isoformat()

print("\nCurrent Date: ",current_date)
print("30 days after current date : ",days_after)
```

Output:

```
Current Date:  2019-11-02
30 days after current date :  2019-12-02
```

n day's before date:

```
from datetime import date, timedelta

current_date = date.today().isoformat()
days_before = (date.today()-timedelta(days=30)).isoformat()

print("\nCurrent Date: ",current_date)
print("30 days before current date: ",days_before)
```

Output:

```
Current Date:  2019-11-02
30 days before current date:  2019-10-03
```

Converting timestamp to date time:

The datetime module can convert a POSIX timestamp to a ITC datetime object.

The Epoch is January 1st, 1970 midnight.

```
import time
from datetime import datetime
seconds_since_epoch=time.time()  #1469182681.709

utc_date=datetime.utcfromtimestamp(seconds_since_epoch)
print(utc_date)
```

Output:

```
2019-11-01 09:53:20.657171
```

Simple date arithmetic:

```
import datetime

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

yesterday = today - datetime.timedelta(days=1)
print('Yesterday:', yesterday)

tomorrow = today + datetime.timedelta(days=1)
print('Tomorrow:', tomorrow)

print('Time between tomorrow and yesterday:', tomorrow - yesterday)
```

Output:

```
Today: 2019-11-01
Yesterday: 2019-10-31
Tomorrow: 2019-11-02
Time between tomorrow and yesterday: 2 days, 0:00:00
```

Subtracting months from a date:

```
import calendar
from datetime import date

def monthdelta(date, delta):
    m, y = (date.month+delta) % 12, date.year + ((date.month)+delta-1) // 12
    if not m: m = 12
    d = min(date.day, calendar.monthrange(y, m)[1])
    return date.replace(day=d,month=m, year=y)

next_month = monthdelta(date.today(), 1) #datetime.date(2019, 10, 23)
print(next_month)
```

Output:

```
2019-12-01
```

Using the dateutils module:

```
import datetime
import dateutil.relativedelta

d = datetime.datetime.strptime("2019-03-31", "%Y-%m-%d")
d2 = d - dateutil.relativedelta.relativedelta(months=1)
#datetime.datetime(2019, 2, 28, 0, 0)
print(d2)
```

Output:

```
2019-02-28 00:00:00
```

Switching between time zones:

To switch between time zones, we need datetime objects that are timezone-aware.

```
from datetime import datetime
from dateutil import tz

utc = tz.tzutc()
local = tz.tzlocal()

utc_now = datetime.utcnow()
utc_now # Not timezone-aware.

utc_now = utc_now.replace(tzinfo=utc)
utc_now # Timezone-aware.

local_now = utc_now.astimezone(local)
local_now # Converted to local time.
print(local_now)
```

Output:

```
2019-11-01 10:10:09.685012+00:00
```

Fuzzy datetime parsing (extracting datetime out of a text):

```
from dateutil.parser import parse

dt = parse("Today is January 1, 2019 at 8:21:00AM", fuzzy=True)
print(dt)
```

Output:

```
2019-01-01 08:21:00
```

Get an ISO 8601 timestamp:

Without timezone, with microseconds:

```
from datetime import datetime

print (datetime.now().isoformat())
```

Output:

```
2019-11-01T10:42:00.720818
```

With timezone, with microseconds:

```
from datetime import datetime
from dateutil.tz import tzlocal

print (datetime.now(tzlocal()).isoformat())
```

Output:

```
2019-11-01T10:46:20.965506+00:00
```

With timezone, without microseconds:

```
from datetime import datetime
from dateutil.tz import tzlocal

print (datetime.now(tzlocal()).replace(microsecond=0).isoformat())
```

Output:

```
2019-11-01T10:49:58+00:00
```

Parsing a string with a short time zone name into a timr zone aware datetime object:

```
from dateutil import tz
from dateutil.parser import parse

ET = tz.gettz('US/Eastern')
CT = tz.gettz('US/Central')
MT = tz.gettz('US/Mountain')
PT = tz.gettz('US/Pacific')

us_tzinfos = {'CST': CT, 'CDT': CT,
              'EST': ET, 'EDT': ET,
              'MST': MT, 'MDT': MT,
              'PST': PT, 'PDT': PT}

dt_est = parse('2018-1-2 04:00:00 EST', tzinfos=us_tzinfos)
dt_pst = parse('2019-3-11 16:00:00 PST', tzinfos=us_tzinfos)
print (dt_est)
print (dt_pst)
```

Output:

```
2018-01-02 04:00:00-05:00
2019-03-11 16:00:00-07:00
```

Parsing an arbitrary ISO 8601 timestamp with minimal libraries:

Python has only limited support for parsing ISO 8601 timestamps and for strptime you need to know exactly what format it is in. The stringification of a datetime is an ISO 8601 timestamp, with space as a separator and 6 digit fraction:

```
import datetime
print (str(datetime.datetime(2019, 7, 22, 9, 25, 59, 555555)))
```

Output:

```
2019-07-22 09:25:59.555555
```

but if the fraction is 0, no fractional part is output

```
import datetime
print(str(datetime.datetime(2019, 7, 22, 9, 25, 59, 0)))
```

Output:

```
2019-07-22 09:25:59.555555
```

Parsing a string into a timezone aware datetime object:

Python 3.2+ has support for %z format when parsing a string into a datetime object.

UTC offset in the form +HHMM or -HHMM (empty string if the object is naive).

```
import datetime
dt = datetime.datetime.strptime("2019-04-15T08:27:18-0500", "%Y-%m-
%dT%H:%M:%S%z")
print(dt)
```

Output:

```
2019-04-15 08:27:18-05:00
```

Fixed Offset Time Zones

```
from datetime import datetime, timedelta, timezone
JST = timezone(timedelta(hours=+9))

dt = datetime(2019, 1, 1, 12, 0, 0, tzinfo=JST)
print(dt)
# 2019-01-01 12:00:00+09:00

print(dt.tzname())
# UTC+09:00

dt = datetime(2019, 1, 1, 12, 0, 0, tzinfo=timezone(timedelta(hours=9),
'JST'))
print(dt.tzname)
# 'JST'
```

Output:

```
2019-01-01 12:00:00+09:00
UTC+09:00
<built-in method tzname of datetime.datetime object at 0x7f91a9437360>
```

Zones with daylight savings time using third party library:

Use the tz.gettz() method to get a time zone object, which can then be passed directly to the datetime constructor:

```
from datetime import datetime
from dateutil import tz
local = tz.gettz() # Local time
PT = tz.gettz('US/Pacific') # Pacific time
```

```
dt_l = datetime(2019, 1, 1, 12, tzinfo=local) # I am in EST
dt_pst = datetime(2019, 1, 1, 12, tzinfo=PT)
dt_pdt = datetime(2019, 7, 1, 12, tzinfo=PT) # DST is handled automatically
print(dt_l)
# 2019-01-01 12:00:00-05:00
print(dt_pst)
# 2019-01-01 12:00:00-08:00
print(dt_pdt)
# 2019-07-01 12:00:00-07:00
```

Output:

```
2019-01-01 12:00:00+00:00
2019-01-01 12:00:00-08:00
2019-07-01 12:00:00-07:00
```

List of the Date format codes:

Directive	**Meaning**	**Example**	**Notes**
%a	Weekday as locale's abbreviated name.	Sun, Mon, …, Sat (en_US); So, Mo, …, Sa (de_DE)	(1)
%A	Weekday as locale's full name.	unday, Monday, …, Saturday (en_US); Sonntag, Montag, …, Samstag (de_DE)	(1)
%w	Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.	0, 1, …, 6	
%d	Day of the month as a zero-padded decimal number.	01, 02, …, 31	(9)
%b	Month as locale's abbreviated name.	Jan, Feb, …, Dec (en_US);	(1)

		Jan, Feb, …, Dez (de_DE)	
%B	Month as locale’s full name.	January, February, …, December (en_US); Januar, Februar, …, Dezember (de_DE)	(1)
%m	Month as a zero-padded decimal number.	01, 02, …, 12	(9)
%y	Year without century as a zero-padded decimal number.	00, 01, …, 99	(9)
%Y	Year with century as a decimal number.	0001, 0002, …, 2013, 2014, …, 9998, 9999	(2)
%H	Hour (24-hour clock) as a zero-padded decimal number.	00, 01, …, 23	(9)
%I	Hour (12-hour clock) as a zero-padded decimal number.	01, 02, …, 12	(9)
%p	Locale’s equivalent of either AM or PM.	AM, PM (en_US); am, pm (de_DE)	(1), (3)
%M	Minute as a zero-padded decimal number.	00, 01, …, 59	(9)
%S	Second as a zero-padded decimal number.	00, 01, …, 59	(4), (9)
%f	Microsecond as a decimal number, zero-padded on the left.	000000, 000001, …, 999999	(5)

%z	UTC offset in the form ±HHMM[SS[.ffffff]] (empty string if the object is naive).	(empty), +0000, -0400, +1030, +063415, -030712.345216	(6)
%Z	Time zone name (empty string if the object is naive).	(empty), UTC, EST, CST	
%j	Day of the year as a zero-padded decimal number.	001, 002, …, 366	(9)
%U	Week number of the year (Sunday as the first day of the week) as a zero padded decimal number. All days in a new year preceding the first Sunday are considered to be in week 0.	00, 01, …, 53	(7), (9)
%W	Week number of the year (Monday as the first day of the week) as a decimal number. All days in a new year preceding the first Monday are considered to be in week 0.	00, 01, …, 53	(7), (9)
%c	Locale’s appropriate date and time representation.	Tue Aug 16 21:30:00 1988 (en_US); Di 16 Aug 21:30:00 1988 (de_DE)	(1)

%x	Locale’s appropriate date representation.	08/16/88 (None);
		08/16/1988 (en_US);
		16.08.1988 (de_DE)
			(1)

%X	Locale’s appropriate time representation.	21:30:00 (en_US); 21:30:00 (de_DE)	(1)
%%	A literal '%' character.	%	