Python datetime module

Last Updated : 26 Jul, 2025

In Python, **date** and **time** are not built-in types but are handled using built-in **datetime** module. This module offers classes to efficiently work with dates, times and intervals, providing many useful methods. Date and DateTime are objects, so manipulating them means working with objects, not plain strings or timestamps.

Why do we need Datetime module?

1. Helps work with dates and times in real-world applications like scheduling or logging.
2. Allows easy calculation of differences between two dates or times.
3. Supports formatting and parsing of date/time strings for user-friendly outputs.
4. Useful for time-stamping events, files or data entries.
5. Essential for handling time zones, durations and calendar-based operations.

Commonly used classes in datetime module

Class	Description
date	Represents a date (year, month, day) in the Gregorian calendar
time	Represents a time independent of any date (hour, minute, second, microsecond, tzinfo)
datetime	Combines date and time (year, month, day, hour, minute, second, microsecond, tzinfo)

Class	Description
timedelta	Represents difference between two dates or times
tzinfo	Abstract base class for timezone information
timezone	Fixed offset timezone class (subclass of tzinfo) introduced in Python 3.2

Date class

The date class provided by datetime module, is used to create and manipulate date objects. When an instance of this class is created, it represents a specific calendar date in ISO 8601 format: **YYYY-MM-DD**.

Syntax

class datetime.date(year, month, day)

Parameters:

- **year:** An integer between MINYEAR (usually 1) and MAXYEAR (usually 9999).
- **month:** An integer from 1 (January) to 12 (December).
- **day:** An integer valid for the specified month and year (e.g., 28 or 29 for February, depending on leap year).

Important Notes

- *Providing invalid types (e.g., string instead of int) raises a TypeError.*
- *Providing out-of-range values raises a ValueError.*
- *The date object does not include time or timezone information — for that, use datetime.depending on leap year).*

Example 1: Creating a Date Object

```
from datetime import date
d = date(1996, 12, 11)
print(d)
```

Output

```
1996-12-11
```

Example 2: Get Current Date

```
from datetime import date
t = date.today()
print(t)
```

Output

```
2025-07-26
```

Example 3: Accessing Year, Month and Day Attributes

```
from datetime import date
t = date.today()
print(t.year)
print(t.month)
print(t.day)
```

Output

```
2025
7
26
```

Example 4: Create Date from Timestamp

```
from datetime import datetime
date_time = datetime.fromtimestamp(1887639468)
print(date_time)
print(date_time.date())
```

Output

```
2029-10-25 16:17:48
2029-10-25
```

Example 5: Convert Date to String

```
from datetime import date
t = date.today()
date_str = t.isoformat()
print(date_str)
print(type(date_str))
```

Output

```
2025-07-26
<class 'str'>
```

Important Date Class Methods

Function Name	Description
ctime()	Return a string representing the date
fromisocalendar()	Returns a date corresponding to the ISO calendar
fromisoformat()	Returns a date object from the string representation of the date
fromordinal()	Returns a date object from the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1

Function Name	Description
fromtimestamp()	Returns a date object from the POSIX timestamp
isocalendar()	Returns a tuple year, week, and weekday
isoformat()	Returns the string representation of the date
isoweekday()	Returns the day of the week as an integer where Monday is 1 and Sunday is 7
replace()	Changes the value of the date object with the given parameter
strftime()	Returns a string representation of the date with the given format
timetuple()	Returns an object of type time.struct_time
today()	Returns the current local date
toordinal()	Return the proleptic Gregorian ordinal of the date, where January 1 of year 1 has ordinal 1
weekday()	Returns the day of the week as integer where Monday is 0 and Sunday is 6

Time class

The time class in Python represents a specific time of day, independent of any particular date. It is used to create time objects that include hour, minute, second, microsecond and timezone information.

Syntax

class datetime.time(hour=0, minute=0, second=0, microsecond=0, tzinfo=None)

All the arguments are optional. tzinfo can be None otherwise all the attributes must be integer in following range -

- 0 <= hour < 24
- 0 <= minute < 60
- 0 <= second < 60
- 0 <= microsecond < 1000000

Example 1: Time object representing time in Python

```
from datetime import time

# Create time object with hour, minute and second
my_time = time(13, 24, 56)
print("Entered time:", my_time)

# Time object with only minute specified
my_time = time(minute=12)
print("Time with one argument:", my_time)

# Time object with default (00:00:00)
my_time = time()
print("Time without argument:", my_time)

# time(hour=26)      → ValueError: hour must be in 0..23
# time(hour='23')    → TypeError: string passed instead of int
```

Output

```
Entered time: 13:24:56
Time with one argument: 00:12:00
```

```
Time without argument: 00:00:00
```

Example 2: Get hours, minutes, seconds and microseconds

After creating a time object, its attributes can also be printed separately.

```
from datetime import time

Time = time(11, 34, 56)
print("hour =", Time.hour)
print("minute =", Time.minute)
print("second =", Time.second)
print("microsecond =", Time.microsecond)
```

Output

```
hour = 11
minute = 34
second = 56
microsecond = 0
```

Example 3: Convert Time object to String

We can convert time object to string using **isoformat()** method.

```
from datetime import time

# Creating Time object
Time = time(12,24,36,1212)

# Converting Time object to string
Str = Time.isoformat()
print("String Representation:", Str)
print(type(Str))
```

Output

```
String Representation: 12:24:36.001212
<class 'str'>
```

List of Time class Methods

Function Name	Description
dst()	Returns tzinfo.dst() is tzinfo is not None
fromisoformat()	Returns a time object from the string representation of the time
isoformat()	Returns the string representation of time from the time object
replace()	Changes the value of the time object with the given parameter
strftime()	Returns a string representation of the time with the given format
tzname()	Returns tzinfo.tzname() is tzinfo is not None
utcoffset()	Returns tzinfo.utcffsets() is tzinfo is not None

Datetime class

The datetime class represents both date and time components in a single object. It combines features of the date and time classes, allowing to work with complete timestamps. It uses Gregorian calendar and assumes 24-hour days with exactly 86,400 seconds.

Syntax

class datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None)

The year, month and day arguments are mandatory. tzinfo can be None, rest all attributes must be an integer in following range -

- MINYEAR <= year <= MAXYEAR
- 1 <= month <= 12
- 1 <= day <= number of days in the given month and year
- 0 <= hour < 24
- 0 <= minute < 60
- 0 <= second < 60
- 0 <= microsecond < 1000000

> ***Note*** *- Passing an argument other than integer will raise a TypeError and passing arguments outside the range will raise ValueError.*

Example 1: DateTime object representing DateTime in Python

```
from datetime import datetime

# Initializing constructor
a = datetime(1999, 12, 12)
print(a)

# Initializing constructor with time parameters as well
a = datetime(1999, 12, 12, 12, 12, 12, 342380)
print(a)
```

Output

```
1999-12-12 00:00:00
1999-12-12 12:12:12.342380
```

Example 2: Get year, month, hour, minute and timestamp

After creating a DateTime object, its attributes can also be printed separately.

```
from datetime import datetime

a = datetime(1999, 12, 12, 12, 12, 12)
print("year =", a.year)
print("month =", a.month)
print("hour =", a.hour)
print("minute =", a.minute)
```

```
print("timestamp =", a.timestamp())
```

Output

```
year = 1999
month = 12
hour = 12
minute = 12
timestamp = 945000732.0
```

Example 3: Current date and time

You can print current date and time using Datetime.now() function. now() function returns current local date and time.

```
from datetime import datetime

# Calling now() function
today = datetime.now()
print("Current date and time is", today)
```

Output

```
Current date and time is 2025-07-26 05:24:55.959180
```

Example 4: Convert Python Datetime to String

We can convert Datetime to string in Python using datetime.strftime and datetime.isoformat methods.

```
from datetime import datetime as dt

# Getting current date and time
now = dt.now()
string = dt.isoformat(now)
print(string)
print(type(string))
```

Output

```
2025-07-26T05:25:45.477917
```

```
<class 'str'>
```

List of Datetime Class Methods

Function Name	Description
astimezone()	Returns the DateTime object containing timezone information.
combine()	Combines the date and time objects and return a DateTime object
ctime()	Returns a string representation of date and time
date()	Return the Date class object
fromisoformat()	Returns a datetime object from the string representation of the date and time
fromordinal()	Returns a date object from the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1. The hour, minute, second, and microsecond are 0
fromtimestamp()	Return date and time from POSIX timestamp
isocalendar()	Returns a tuple year, week, and weekday
isoformat()	Return the string representation of date and time

Function Name	Description
isoweekday()	Returns the day of the week as integer where Monday is 1 and Sunday is 7
now()	Returns current local date and time with tz parameter
replace()	Changes the specific attributes of the DateTime object
strftime()	Returns a string representation of the DateTime object with the given format
strptime()	Returns a DateTime object corresponding to the date string
time()	Return the Time class object
timetuple()	Returns an object of type time.struct_time
timetz()	Return the Time class object
today()	Return local DateTime with tzinfo as None
toordinal()	Return the proleptic Gregorian ordinal of the date, where January 1 of year 1 has ordinal 1
tzname()	Returns the name of the timezone
utcfromtimestamp()	Return UTC from POSIX timestamp
utcoffset()	Returns the UTC offset

Function Name	Description
utcnow()	Return current UTC date and time
weekday()	Returns the day of the week as integer where Monday is 0 and Sunday is 6

Timedelta Class

Python timedelta class is used for calculating differences in dates and also can be used for date manipulations in Python. It is one of easiest ways to perform date manipulations.

Syntax

class datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)

All parameters are optional and can be used in any combination

Example 1: Add days to DateTime object

The timedelta function demonstration

```
from datetime import datetime, timedelta

# Get the current date and time
now = datetime.now()
print("Current Date & Time:", now)

# Add 2 years (approx. 730 days)
after_2_years = now + timedelta(days=730)
print("After 2 Years:", after_2_years)

# Add 2 days
after_2_days = now + timedelta(days=2)
print("After 2 Days:", after_2_days)
```

Output

```
Current Date & Time: 2025-07-26 05:38:58.395824
After 2 Years: 2027-07-26 05:38:58.395824
After 2 Days: 2025-07-28 05:38:58.395824
```

Example 2: Difference between two date and times

Date and Time differences can also be found using this class.

```python
from datetime import datetime, timedelta

# Current date and time
now = datetime.now()
print("Current Time:", now)

# New time after 2 days
new_time = now + timedelta(days=2)
print("New Time (+2 days):", new_time)

# Time difference
print("Time Difference:", new_time - now)
```

Output

```
Current Time: 2025-07-26 05:40:17.572931
New Time (+2 days): 2025-07-28 05:40:17.572931
Time Difference: 2 days, 0:00:00
```

Operations supported by Timedelta Class

Operator	Description
Addition (+)	Adds and returns two timedelta objects
Subtraction (-)	Subtracts and returns two timedelta objects
Multiplication (*)	Multiplies timedelta object with float or int

Operator	Description
Division (/)	Divides the timedelta object with float or int
Floor division (//)	Divides the timedelta object with float or int and return the int of floor value of the output
Modulo (%)	Divides two timedelta object and returns the remainder
+(timedelta)	Returns the same timedelta object
-(timedelta)	Returns the resultant of -1*timedelta
abs(timedelta)	Returns the +(timedelta) if timedelta.days > 1=0 else returns -(timedelta)
str(timedelta)	Returns a string in the form (+/-) day[s], HH:MM:SS.UUUUUU
repr(timedelta)	Returns the string representation in the form of the constructor call

Tzinfo class

The tzinfo class in Python is an abstract base for handling time zone info in datetime objects. It can't be used directly instead, subclasses or libraries provide actual time zone support for accurate date-time calculations.

> ***Note:*** *tzinfo is an abstract base class, so it is not meant to be instantiated directly. Instead, create a subclass of tzinfo and implement its required methods like utcoffset(), dst() and tzname().*

Example:

This example shows how to get system time zone and convert a naive datetime into an aware one using **dateutil.tz.gettz()** with "Europe/Berlin" time zone.

```
import datetime as dt
from dateutil import tz

# Get system's local time zone name
tz_string = dt.datetime.now(dt.timezone.utc).astimezone().tzname()
print("System Time Zone:", tz_string)

# Assigning timezone using dateutil
berlin = tz.gettz('Europe/Berlin')
dt1 = dt.datetime(2022, 5, 21, 12, 0)
dt2 = dt.datetime(2022, 12, 21, 12, 0, tzinfo=berlin)

print("Naive Object:", dt1.tzname())
print("Aware Object:", dt2.tzname())
```

Output

```
System Time Zone: UTC
Naive Object: None
Aware Object: CET
```

Explanation:

- **dt.datetime.now(dt.timezone.utc).astimezone().tzname():** gets system's current time zone name by converting UTC time to local time zone.
- **berlin = tz.gettz('Europe/Berlin'):** fetches time zone information for Europe/Berlin.
- **tzname()** returns name of time zone for a given datetime object

List of Python DateTime.tzinfo() Objects

Function Name	Description
dst()	Returns tzinfo.dst() is tzinfo is not None

Function Name	Description
fromutc()	The purpose of this function is to adjust the date time data, returning an equivalent DateTime in self's local time.
tzname()	Returns tzinfo.tzname() is tzinfo is not None
utcoffset()	Returns tzinfo.utcffsets() is tzinfo is not None

Timezone class

Timezones in Python's datetime module help display time for different regions. The pytz module allows easy conversion between timezones, making it useful for apps with international users.

Syntax

timezone(offset, name=None)

Parameter:

- **offset:** A timedelta object (e.g., timedelta(hours=5, minutes=30)).
- **name:** Optional string name for the timezone (e.g., "IST").

Example:

This code shows how to get current UTC time and convert it to different time zones using pytz module. It prints time in UTC, Asia/Kolkata, Europe/Kiev and America/New_York using a consistent format.

```
from datetime import datetime
from pytz import timezone

format = "%Y-%m-%d %H:%M:%S %Z%z"
```

```
now_utc = datetime.now(timezone('UTC')) # Current time in UTC
print(now_utc.strftime(format))

timezones = ['Asia/Kolkata', 'Europe/Kiev', 'America/New_York']
for tzone in timezones:
    now_asia = now_utc.astimezone(timezone(tzone))  # Convert to Asia/Ko
zone
    print(now_asia.strftime(format))
```

Output

```
2025-07-26 06:25:28 UTC+0000
2025-07-26 11:55:28 IST+0530
2025-07-26 09:25:28 EEST+0300
2025-07-26 02:25:28 EDT-0400
```

Explanation:

- **"%Y-%m-%d %H:%M:%S %Z%z":** Defines format for displaying date and time including time zone name (%Z) and UTC offset (%z).
- **datetime.now(timezone('UTC')):** Gets current date and time in UTC by attaching UTC time zone to current datetime.
- **now_utc.strftime(format):** prints current UTC time formatted according to specified pattern.
- **now_utc.astimezone(timezone(tzone)):** converts UTC time to current time zone (tzone).

Next Article

Python Introduction

Similar Reads

Python Tutorial - Learn Python Programming Language

Python is one of the most popular programming languages. Itâ€™s simple to use, packed with features and supported by a wide range of libraries and frameworks. Its clean syntax makes it beginner-friendly. It'sA high...

10 min read

Python Fundamentals

Python Data Structures

Advanced Python

Data Science with Python

Web Development with Python

Python Practice

GeeksforGeeks
Corporate & Communications Address:
A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:
K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305