Python Datetime Module

The Datetime module allows us to work with date and time objects. It provides three additional data types: `date`, `time` and `datetime`.

```
import datetime
```

date()

```
datetime.date(year: int, month: int, day: int)
```

The date method return a date object with the `year`, `month` and `day` attributes:

```
>>> from datetime import date
>>> obj = date(2022, 12, 1)
>>> obj.year
# 2022
>>> obj.month
# 12
>>> obj.day
# 1
```

time()

```
datetime.time(hour: int, minute: int, second: int)
```

The `time` method return a time object with the `hour`, `minute`, `second`, `microsecond` and `tzinfo` attributes:

```
>>> from datetime import time
>>> obj = time(10, 20, 33)
>>> obj.hour
# 10
>>> obj.second
# 33
>>> obj.microsecond
# 0
```

datetime()

```
datetime.datetime(year, month, day, hour, minute, second)
```

The `datetime` returns an object with both, the `date` and `time` objects attributes:

```
>>> from datetime import datetime
>>> obj = datetime(2024, 12, 1, 15, 35, 59)
>>> obj.year
# 2024
>>> obj.month
# 12
>>> obj.day
# 1
>>> obj.hour
# 15
>>> obj.second
# 59
```

now() and today()

`now` and `today` methods return a `datetime` object with system's exact day and time:

```
>>> from datetime import datetime
>>> now = datetime.now()
```

```
>>> now
# datetime.datetime(2022, 7, 23, 19, 56, 49, 589806)
```

Because the object returned is a `datetime`, we can access both, `date` and `time` attributes:

```
>>> now.date()
# datetime.date(2022, 7, 23)
>>> now.time()
# datetime.time(19, 56, 49, 589806)
>>> now.year
# 2022
>>> now.month
# 7
>>> now.day
# 23
>>> now.hour
# 19
>>> now.minute
# 56
>>> now.second
# 49
>>> now.microsecond
# 589806
```

Additionally, `now` can take a **timezone** object as an optional parameter:

```
>>> from datetime import datetime, timezone
>>> datetime.now(timezone.utc)
# datetime.datetime(2022, 7, 24, 0, 20, 8, 265634, tzinfo=datetime.ti
```

If a *timezone* parameter is not specified, `now` will default to the system timezone.

strftime() and strptime()

You can easily transform between strings and datetime objects with the `strftime` and `strptime` methods.

strftime()

`strftime` allow us to create human formatted strings out of a Python datetime object:

```
>>> from datetime import datetime
>>> now = datetime.now()
>>> now
# datetime.datetime(2022, 7, 23, 20, 31, 19, 751479)

>>> now.strftime("%d-%b-%Y")
# '23-Jul-2022'

>>> now.strftime("%d-%m-%Y")
# '23-07-2022'

>>> now.strftime("%d-%b-%Y")
# '23-Jul-2022'

>>> now.strftime("%d-%m-%Y")
# '23-07-2022'

>>> now.strftime("%m/%d/%Y")
# '07/23/2022'

>>> now.strftime("%b/%d/%Y - %H:%M:%S")
# 'Jul/23/2022 - 20:31:19'
```

You may find the strings passed to `strftime` to be a little strange, but it is pretty easy to understand its meaning. For example, `%m/%d/%Y` will return the month, day, and year separated by `/` (07/23/2022).

strptime()

The `strptime` method creates a `datetime` object from a string.

This method accepts two parameters:

```
obj.strptime(datetime_string, format)
```

- A string representing a datetime object.

- The python format code equivalent to that string.

```
>>> from datetime import datetime

>>> datetime_str = '12-Jul-2023'
>>> datetime.strptime(datetime_str, '%d-%b-%Y')
# datetime.datetime(2023, 7, 12, 0, 0)

>>> datetime_str = 'Jul/12/2023 - 14:38:37'
>>> datetime.strptime(datetime_str, "%b/%d/%Y - %H:%M:%S")
# datetime.datetime(2023, 7, 12, 14, 38, 37)
```

Format Codes

Directive	Meaning	Example
`%a`	Weekday as locale's abbreviated name.	Sun, Mon, …, Sat (en_US)
`%A`	Weekday as locale's full name.	Sunday, Monday, …, Saturday (en_US)
`%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
`%b`	Month as locale's abbreviated name.	Jan, Feb, …, Dec (en_US)
`%B`	Month as locale's full name.	January, February, …, December (en_US)
`%m`	Month as a zero-padded decimal number.	01, 02, …, 12
`%y`	Year without century as a zero-padded decimal number.	00, 01, …, 99
`%Y`	Year with century as a decimal number.	0001, 0002, …, 2013, 2014, …, 9998, 9999
`%H`	Hour (24-hour clock) as a zero-padded decimal number.	00, 01, …, 23

Directive	Meaning	Example
`%I`	Hour (12-hour clock) as a zero-padded decimal number.	01, 02, …, 12
`%p`	Locale's equivalent of either AM or PM.	AM, PM (en_US)
`%M`	Minute as a zero-padded decimal number.	00, 01, …, 59
`%S`	Second as a zero-padded decimal number.	00, 01, …, 59
`%f`	Microsecond as a decimal number, zero-padded on the left.	000000, 000001, …, 999999
`%z`	UTC offset in the form `±HHMM[SS[.ffffff]]` (empty string if the object is naive).	(empty), +0000, -0400, +1030, +063415, -030712.345216
`%Z`	Time zone name (empty string if the object is naive).	(empty), UTC, GMT
`%j`	Day of the year as a zero-padded decimal number.	001, 002, …, 366
`%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
`%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
`%c`	Locale's appropriate date and time representation.	Tue Aug 16 21:30:00 1988 (en_US)
`%x`	Locale's appropriate date representation.	08/16/88 (None)
`%X`	Locale's appropriate time representation.	21:30:00 (en_US)
`%%`	A literal `'%'` character.	%

timedelta()

The `timedelta` object represents the difference between two dates or times.

```
>>> from datetime import datetime

>>> date_1 = datetime.strptime('12-Jul-2023', '%d-%b-%Y')
>>> date_2 = datetime.strptime('01-Jan-2024', '%d-%b-%Y')

>>> difference = date_2 - date_1
>>> difference
# datetime.timedelta(days=173)
>>> difference.days
# 173
```

`timedelta` can add `days`, `seconds` and `microseconds` to a datetime object:

```
>>> from datetime import datetime, timedelta

>>> now = datetime.now()
>>> now
# datetime.datetime(2022, 7, 23, 21, 25, 2, 341081)

>>> now + timedelta(days=10, seconds=15)
# datetime.datetime(2022, 8, 2, 21, 25, 17, 341081)
```

And can subtract `days`, `seconds` and `microseconds` to a datetime object:

```
>>> from datetime import datetime, timedelta

>>> now = datetime.now()
>>> now
# datetime.datetime(2022, 7, 23, 21, 25, 2, 341081)

>>> now - timedelta(days=10, seconds=15)
# datetime.datetime(2022, 7, 13, 21, 59, 41, 100883)
```

Previous page
Copy

Next page
Itertools

Subscribe to pythoncheatsheet.org

Join **8.800+ Python developers** in a two times a month and bullshit free **publication**, full of interesting, relevant links.

Your email address

SUBSCRIBE

Edit this page on GitHub

Do you have a question? ask the community

Do you see a bug? open an issue on GitHub