Python datetime

Python has a module named `datetime` to work with dates and times.

It provides a variety of classes for representing and manipulating dates and times, as well as for formatting and parsing dates and times in a variety of formats.

Example 1: Get Current Date and Time

```
import datetime

# get the current date and time
now = datetime.datetime.now()

print(now)
```

Output

```
2022-12-27 08:26:49.219717
```

Here, we have imported the `datetime` module using the `import datetime` statement.

One of the classes defined in the `datetime` module is the `datetime` class.

We then used the `now()` method to create a `datetime` object containing the current local date and time.

Example 2: Get Current Date

```
import datetime

# get current date
current_date = datetime.date.today()

print(current_date)
```

Output

```
2022-12-27
```

In the above example, we have used the `today()` method defined in the `date` class to get a `datetime` object containing the current local date.

Attributes of datetime Module

We can use the dir() function to get a list containing all attributes of a module.

```
import datetime

print(dir(datetime))
```

Output

```
['MAXYEAR', 'MINYEAR', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_divide_and_round', 'date', 'datetime', 'datetim
```

Among all the attributes of `datetime` module, the most commonly used classes in the `datetime` module are:

- `datetime.datetime` - represents a single point in time, including a date and a time.
- `datetime.date` - represents a date (year, month, and day) without a time.
- `datetime.time` - represents a time (hour, minute, second, and microsecond) without a date.
- `datetime.timedelta` - represents a duration, which can be used to perform arithmetic with `datetime` objects.

Python datetime.date Class

In Python, we can instantiate `date` objects from the `date` class. A date object represents a date (year, month and day).

Example 3: Date object to represent a date

```
import datetime

d = datetime.date(2022, 12, 25)
print(d)
```

Output

```
2022-12-25
```

Here, `date()` in the above example is a constructor of the `date` class. The constructor takes three arguments: `year`, `month` and `day`.

Import Only date Class

We can only import the `date` class from the `datetime` module. For example,

```
from datetime import date

d = date(2022, 12, 25)
print(d)
```

Output

```
2022-12-25
```

Here, `from datetime import date` only imports the `date` class from the `datetime` module.

Example 4: Get the current date using today()

We can create a `date` object containing the current date by using the class method named `today()`. For example,

```
from datetime import date

# today() to get current date
todays_date = date.today()

print("Today's date =", todays_date)
```

Output

```
Today's date = 2022-12-27
```

Example 5: Get the date from a timestamp

We can also create `date` objects from a timestamp.

A UNIX timestamp is the number of seconds between a particular date and **January 1, 1970,** at UTC. You can convert a timestamp to a date using the `fromtimestamp()` method.

```
from datetime import date

timestamp = date.fromtimestamp(1326244364)
print("Date =", timestamp)
```

Output

```
Date = 2012-01-11
```

Example 6: Print today's year, month and day

We can get year, month, day, day of the week, etc. from the `date` object easily. For example,

```
from datetime import date

# date object of today's date
today = date.today()

print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```

Output

```
Current year: 2022
Current month: 12
Current day: 27
```

Python datetime.time Class

A time object instantiated from the `time` class represents the local time.

Example 7: Time object to represent time

```
from datetime import time

# time(hour = 0, minute = 0, second = 0)
a = time()
print(a)

# time(hour, minute and second)
b = time(11, 34, 56)
print(b)

# time(hour, minute and second)
c = time(hour = 11, minute = 34, second = 56)
print(c)

# time(hour, minute, second, microsecond)
d = time(11, 34, 56, 234566)
print(d)
```

Output

```
a = 00:00:00
b = 11:34:56
c = 11:34:56
d = 11:34:56.234566
```

Example 8: Print hour, minute, second and microsecond

Once we create the `time` object, we can easily print its attributes such as `hour`, `minute`, etc. For example,

```
from datetime import time

a = time(11, 34, 56)

print("Hour =", a.hour)
print("Minute =", a.minute)
print("Second =", a.second)
print("Microsecond =", a.microsecond)
```

Output

```
Hour = 11
Minute = 34
Second = 56
Microsecond = 0
```

Here, notice that we haven't passed the *microsecond* argument. Hence, its default value **0** is printed.

The datetime.datetime Class

The `datetime` module has a class named `datetime` that can contain information from both `date` and `time` objects.

Example 9: Python datetime object

```
from datetime import datetime

# datetime(year, month, day)
a = datetime(2022, 12, 28)
print(a)

# datetime(year, month, day, hour, minute, second, microsecond)
b = datetime(2022, 12, 28, 23, 55, 59, 342380)
print(b)
```

Output

```
2022-12-28 00:00:00
2022-12-28 23:55:59.342380
```

The first three arguments *year*, *month* and *day* in the `datetime()` constructor are mandatory.

Example 10: Print year, month, hour, minute and timestamp

```
from datetime import datetime

a = datetime(2022, 12, 28, 23, 55, 59, 342380)

print("Year =", a.year)
print("Month =", a.month)
print("Hour =", a.hour)
print("Minute =", a.minute)
print("Timestamp =", a.timestamp())
```

Output

```
year = 202
month = 12
day = 28
hour = 23
minute = 55
timestamp = 1511913359.34238
```

Python datetime.timedelta Class

A `timedelta` object represents the difference between two dates or times. For example,

```
from datetime import datetime, date

# using date()
t1 = date(year = 2018, month = 7, day = 12)
t2 = date(year = 2017, month = 12, day = 23)

t3 = t1 - t2

print("t3 =", t3)

# using datetime()
t4 = datetime(year = 2018, month = 7, day = 12, hour = 7, minute = 9, second = 33)
t5 = datetime(year = 2019, month = 6, day = 10, hour = 5, minute = 55, second = 13)
t6 = t4 - t5
print("t6 =", t6)

print("Type of t3 =", type(t3))
print("Type of t6 =", type(t6))
```

Output

```
t3 = 201 days, 0:00:00
t6 = -333 days, 1:14:20
Type of t3 = <class 'datetime.timedelta'>
Type of t6 = <class 'datetime.timedelta'>
```

Notice, both *t3* and *t6* are of `<class 'datetime.timedelta'>` type.

Example 12: Difference between two timedelta objects

```
from datetime import timedelta

t1 = timedelta(weeks = 2, days = 5, hours = 1, seconds = 33)
t2 = timedelta(days = 4, hours = 11, minutes = 4, seconds = 54)

t3 = t1 - t2

print("t3 =", t3)
```

Output

```
t3 = 14 days, 13:55:39
```

Here, we have created two `timedelta` objects *t1* and *t2*, and their difference is printed on the screen.

Example 14: Time duration in seconds

We can get the total number of seconds in a `timedelta` object using the `total_seconds()` method.

```
from datetime import timedelta

t = timedelta(days = 5, hours = 1, seconds = 33, microseconds = 233423)
print("Total seconds =", t.total_seconds())
```

Output

```
Total seconds = 435633.233423
```

Python format datetime

The way date and time are represented may be different in different places, organizations, etc. It's more common to use `mm/dd/yyyy` in the US, whereas `dd/mm/yyyy` is more common in the UK.

Python has `strftime()` and `strptime()` methods to handle this.

Python strftime() Method

The `strftime()` method is defined under classes `date`, `datetime` and `time`. The method creates a formatted string from a given `date`, `datetime` or `time` object.

Let's see an example.

```
from datetime import datetime

# current date and time
now = datetime.now()

t = now.strftime("%H:%M:%S")
print("Time:", t)

s1 = now.strftime("%m/%d/%Y, %H:%M:%S")
# mm/dd/YY H:M:S format
print("s1:", s1)
```

```
s2 = now.strftime("%d/%m/%Y, %H:%M:%S")
# dd/mm/YY H:M:S format
print("s2:", s2)
```

Output

```
time: 04:34:52
s1: 12/26/2018, 04:34:52
s2: 26/12/2018, 04:34:52
```

Here, `%Y`, `%m`, `%d`, `%H` etc. are format codes. The `strftime()` method takes one or more format codes and returns a formatted string based on it.

In the above example, *t*, *s1* and *s2* are strings.

- `%Y` - year [0001,..., 2018, 2019,..., 9999]
- `%m` - month [01, 02, ..., 11, 12]
- `%d` - day [01, 02, ..., 30, 31]
- `%H` - hour [00, 01, ..., 22, 23
- `%M` - minute [00, 01, ..., 58, 59]
- `%S` - second [00, 01, ..., 58, 59]

To learn more about `strftime()` and format codes, visit: Python strftime().

Python strptime() Method

The `strptime()` method creates a `datetime` object from a given string (representing date and time). For example,

```
from datetime import datetime

date_string = "25 December, 2022"
print("date_string =", date_string)

# use strptime() to create date object
date_object = datetime.strptime(date_string, "%d %B, %Y")

print("date_object =", date_object)
```

Output

```
date_string = 25 december, 2022
date_object = 2018-06-21 00:00:00
```

The `strptime()` method takes two arguments:

- a string representing date and time
- format code equivalent to the first argument

By the way, `%d`, `%B` and `%Y` format codes are used for *day*, *month*(full name) and *year* respectively.

To learn more about `strptime()` and format codes, visit: Python strptime().

Handling timezone in Python

Suppose, we are working on a project and need to display date and time based on their timezone.

Rather than trying to handle the timezone yourself, we suggest using a third-party pytZ module.

```
from datetime import datetime
import pytz

local = datetime.now()
print("Local:", local.strftime("%m/%d/%Y, %H:%M:%S"))


tz_NY = pytz.timezone('America/New_York')
datetime_NY = datetime.now(tz_NY)
print("NY:", datetime_NY.strftime("%m/%d/%Y, %H:%M:%S"))

tz_London = pytz.timezone('Europe/London')
datetime_London = datetime.now(tz_London)
print("London:", datetime_London.strftime("%m/%d/%Y, %H:%M:%S"))
```

Output

```
Local: 12/27/2022, 09:40:19
NY: 12/27/2022, 04:40:19
London: 12/27/2022, 09:40:19
```

Here, *datetime_NY* and *datetime_London* are `datetime` objects containing the current date and time of their respective timezone.

Also Read:

- Python get current time
- How to get current date and time in Python
- Python time module

Table of Contents

- Introduction
- Example 1: Get Current Date and Time
- Attributes of datetime Module
- Python datetime.date Class
- Example 3: Date object to represent a date
- Example 5: Get the date from a timestamp
- Example 6: Print today's year, month and day
- Python datetime.time Class
- Example 7: Time object to represent time
- Example 8: Print hour, minute, second and microsecond
- The datetime.datetime Class
- Example 9: Python datetime object
- Example 10: Print year, month, hour, minute and timestamp
- Python datetime.timedelta Class
- Example 12: Difference between two timedelta objects
- Python format datetime
- Python strftime() Method
- Python strptime() Method
- Handling timezone in Python