Datetime Module Quick Reference

The Python \color{red}{\verb|datetime|}datetime module is a relatively complex module that simplifies operations on dates and times. This page gives a quick reference for some of the basic functionality of the module. For the curious, you may want to look at the full [documentation](https://docs.python.org/3/library/datetime.html).

Time Zones

When you write a script to compare dates and times, you often have to worry about time zones, as the times may be from different parts of the world. For our purposes, we are going to ignore time zones and just work with dates. This simplifies a lot of the processing. And, for the most part, you do not need to worry about time zones when you are only dealing with dates. The one exception is that when you use the module to get today's date the answer might be different depending on what timezone the computer is in.

In the Python documentation of the \color{red}{\verb|datetime|}datetime module, dates without an associated time zone are called "naive" dates. An "aware" date or time object does have an associated time zone. For our purposes, we will limit ourselves to using naive dates.

Date Objects

One of the primary features of the \color{red}{\verb|datetime|}datetime module is the \color{red}{\verb|date|}date type. A date object represents a date (year, month, and day). To create a date object, you use the function \color{red}{\verb|datetime.date(year, month, day)|}datetime.date(year, month, day). The three inputs, as the names imply, are the year, month, and day of the date you would like to create. The \color{red}{\verb|datetime|}datetime module defines two constants (\color{red}{\verb|datetime.MINYEAR|}(datetime.MINYEAR and \color{red}{\verb|datetime.MAXYEAR|})datetime.MAXYEAR) that indicate the range of valid years. The inputs to the \color{red}{\verb|datetime.date|}datetime.date function must meet the following conditions:

* \color{red}{\verb|datetime.MINYEAR|}datetime.MINYEAR \le≤ year \le≤ \color{red}{\verb|datetime.MAXYEAR|}datetime.MAXYEAR
* 1 \le1≤ month \le 12≤12
* 1 \le1≤ day \le≤ number of days in the given month and year

There will be an error If not all inputs are provided or they do not fall in the given ranges.

A date object has three attributes that you can access in order to determine the year, month, and day. These attributes have the obvious names (\color{red}{\verb|year|}year, \color{red}{\verb|month|}month, and \color{red}{\verb|day|}day).

Here is a simple example:

import datetime

date1 = datetime.date(2016, 1, 7)

print(date1)

print(date1.year)

print(date1.month)

print(date1.day)

As the example shows, you can also print date objects directly. They will display in a human readable format.

Comparisons with Dates

As with many Python data types, you can compare two dates to each other. You can use all six of the Python comparison operators (\color{red}{\verb|==|}==, \color{red}{\verb|!=|}!=, \color{red}{\verb|>=|}>=, \color{red}{\verb|<=|}<=, \color{red}{\verb|>|}>, \color{red}{\verb|<|}<) and they have their obvious meanings. These comparison operations, as always, evaluate to boolean values, so they evaluate to either \color{red}{\verb|True|}True or \color{red}{\verb|False|}False.

Here is a simple example:

import datetime

date1 = datetime.date(2006, 3, 15)

date2 = datetime.date(2009, 4, 30)

print(date1 != date2)

print(date1 > date2)

Arithmetic with Dates

Dates support some limited arithmetic operations. While it doesn't make any sense to add two dates together (what would that mean?), it does make sense to subtract two dates. When you subtract two dates, you would expect the answer to be the time between those dates, which is exactly what happens. When you subtract two dates, the result is a \color{red}{\verb|timedelta|}timedelta object. This is the difference (or delta) between two dates. You can print a timedelta object directly, and it will display in a human readable format.

However, more likely, you need to do some computation with the timedelta. A timedelta object has three attributes: \color{red}{\verb|days|}days, \color{red}{\verb|seconds|}seconds, and \color{red}{\verb|microseconds|}microseconds. These attributes allow you to determine the exact amount of the time difference. When you subtract two dates, seconds and microseconds will always be 0 as the result is an integral number of days.

While you cannot add two dates, you can add a timedelta to a date. This evaluates to a new date that is different from the original date by the difference represented by the timedelta.

Here is a simple example of arithmetic with dates:

import datetime

date1 = datetime.date(2006, 3, 15)

date2 = datetime.date(2009, 4, 30)

difference = date2 - date1

print(difference)

print(difference.days)

date3 = date1 + difference

print(date2 == date3)

Today

It is useful to be able to get a date object that represents today's date. You could create a new date object, giving it today's year, month, and day, but that would require you to already know when today is within your program! To solve this problem, the \color{red}{\verb|datetime|}datetime module has a function, \color{red}{\verb|datetime.date.today()|}datetime.date.today(), that returns a date object representing today's date (where "today" is the day on which the program is run).

Here is an example:

import datetime

todays\_date = datetime.date.today()

print(todays\_date)

The function \color{red}{\verb|datetime.date.today()|}datetime.date.today() returns a date object, so you can access the year, month, and day (for example, \color{red}{\verb|todays\_date.year|}todays\_date.year, \color{red}{\verb|todays\_date.month|}todays\_date.month, and \color{red}{\verb|todays\_date.day|}todays\_date.day) just as you would with any other date object, as described above.