Datetime (Sophie)

Python Reference Document for datetime: <https://docs.python.org/3/library/datetime.html?highlight=datetime#module-datetime>

**What is the datetime module?**

Datetime is a built in module in python that makes working with dates and times easier to deal with. Dates and times can be difficult to deal with in programming, especially since different programming languages deal with them in different ways. Add in the need to track time zones if you are working internationally and you end up with another layer of difficulty. The datetime module is here to help you format dates, take dates and times as inputs, and output dates in easy and clear ways.

**How computers count time:**

Nearly all computers count time from an instant called the Unix epoch. This occurred on January 1, 1970, at 00:00:00 UTC. UTC stands for Coordinated Universal Time and refers to the time at a longitude of 0°. UTC is often also called Greenwich Mean Time, or GMT. UTC is not adjusted for daylight saving time, so it consistently keeps twenty-four hours in every day. Computers don’t see time the same way we do. It can take time to wrap one’s head around how computers interpret time, but the datetime module will help you get started.

**Using Datetime in your code:**

As with most modules, datetime will need to be imported. To do this you simply need to add the following statement to the top of your code:

from datetime import date, time, datetime

To get a feel for how this would look in code you can try something like

from datetime import date, time, datetime

date(year=2020, month=1, day=31)

time(hour=13, minute=14, second=31)

datetime(year=2020, month=1, day=31, hour=13, minute=14, second=31)

If you run this code above in a terminal you will end up with something like the following:

>>> from datetime import date, time, datetime

>>> date(year=2020, month=1, day=31)

datetime.date(2020, 1, 31)

>>> time(hour=13, minute=14, second=31)

datetime.time(13, 14, 31)

>>> datetime(year=2020, month=1, day=31, hour=13, minute=14, second=31)

datetime.datetime(2020, 1, 31, 13, 14, 31)

This can be a lot to type, and is definitely not the most efficient way to get what you need. Luckily datetime has some handy built-in functions. I will list a few that you should try in your terminal.

1. date.today()
   1. This creates a datetime.date instance with the current local date
2. datetime.now()
   1. Created a datetime.datetime instance with the current local time
3. datetime.combine()
   1. Combines instances of the two previous commands into a single datetime.datetime instance

These pieces of code can be handy to track things in your future programs. You can easily record the date and time information is entered by a user without having to ask them to put in the date. You can take a user input such as date of birth and convert that into different formats. The uses are quite broad. Datetime is especially useful since all user input is initially stored as strings. You can use commands in the datetime module to convert those strings into data that you can work with.

For example, let's say we have the following string:

2020-01-31

We want to convert that into a datetime.date instance. We would run the following code:

>>> from datetime import datetime

>>> date\_string = "12-20-2016 14:45:37"

>>> format\_string = "%m-%d-%Y %H:%M:%S"

>>> datetime.strptime(date\_string, format\_string)

datetime.datetime(2016, 12, 20, 14, 45, 37)

First we must tell the code the format we are expecting the string to be in. In this case, format\_string = "%m-%d-%Y %H:%M:%S". You can see that the code returns a datetime.date instance storing it as (month, day, year, hour, minute, second) which can now be manipulated with other datetime functions within your code.

Some other useful functions for datetime are as follows:

**Customize Date Formats**

You can customize the format of dates by defining the date formats using strftime method. It converts date objects to strings.

For all of the following formatting assume dt = 27-03-2021

dt.strftime("%d-%m-%Y")

Output

27-03-2021

%d refers to day of the month. In 20-10-2019, %d returns 20.

%m refers to month of the year. In 20-10-2019, %m returns 10.

%Y refers to year. The letter 'Y' is in upper case. In 20-10-2019, %Y returns 2019.

%y refers to year in two-digit format. In 20-10-2019, %y returns 19.

Other popular format codes

* %a returns the first three letter of the weekday Sun
* %A returns the complete name of the weekday Sunday
* %b returns the first three letters of the month Oct
* %B returns the complete name of the month October

dt.strftime("%d/%m/%Y")

27/10/2021

dt.strftime("%b %d, %Y")

Mar 27, 2021

dt.strftime("%B %d, %Y")

March 27, 2021

dt.strftime("%a %B %d, %Y")

Sat March 27, 2021

dt.strftime("%A %B %d, %Y")

Saturday March 27, 2021

dt.strftime("%A, %B %d, %Y")

Saturday, March 27, 2021

**Time**

Time values are defined with datetime.time class. It follows the syntax as shown below -

datetime.time(hour, minute, second, microseconds)

t = datetime.time(21, 2, 3)

print(t)

21:02:03

t.hour will return the hour value

21

t.minute returns the minute value

2

t.second returns the second value

3

t.microsecond returns the microsecond value

0

Pretty straightforward so far, right?

If you want to return the AM/PM values instead of working off of a 24 hour clock you can use the following formats

* %I converts 24 hour time format to 12 hour format.
* %p returns AM PM based on time values.
* %H returns hours of the time value.
* %M returns minutes of the time value.
* %S returns seconds of the time value.

An example would be:

t.strftime("%I:%M %p")

09:02 PM

Lastly, at least for this study guide, there is the timedelta function. This allows you to find the differences between different dates. This is fairly simple once you have your dates or times converted into similar formats using the statements shown above. An example of this would be as follows:

>>> date1 = datetime.date(2021, 3, 27)

>>> date2 = datetime.date(2019, 12, 25)

>>> diff = date1 - date2

>>> diff.days

**Output**

458

The first two statements are assigning your desired dates into variables using datetime.date instances. Line 3 is a simple mathematical statement which subtracts the two dates. The fourth line is the important one. This converts the result into the number of days different. Otherwise you would end up with something odd like:

>>> print(diff)

458 days, 0:00:00

This output is technically correct, but is not as appealing of an output.

Datetime if a very useful module, but covers a broad range of information. This study guide is just a starting point. For more in-depth information I recommend <https://www.listendata.com/2019/07/how-to-use-datetime-in-python.html>. This site goes over multiple uses of the datetime module with examples in a similar manner to what I have shown above.

Example questions:

1. What will print(diff) show once the following code is run?

>>> date1 = datetime.date(2021, 12, 4)

>>> date2 = datetime.date(2016, 12, 20)

>>> diff = date1 - date2

1. 1800 days, 0:00:00
2. 1810 days, 0:00:00
3. 1810 days
4. 1800 days
5. Given dt = “2016-12-4", what code would result in an output of 'December, 04, 2021'
6. dt.strftime("%B, %d, %Y")
7. dt.strftime("%A %B %d, %Y")
8. dt.strftime("%d/%m/%Y")
9. dt.strftime("%A, %B %d, %Y")

Author: Sophie Solo, CSC 121 Study Guide, Datetime module