Theory: Datetime parsing and formatting

28 minutes 0 / 5 problems solved

Around the world, dates are formatted in a variety of ways. Some countries use dots as delimiters, some use slashes. In some date formats, month precedes the date, in others, vice versa. Then, there is always an option to write out dates in words, not just numbers.

It's not surprising then that programming languages have tools for working with different date formats. In this topic, we'll see how this can be done in Python.

If you recall, Python has a built-in module for working with date — `datetime`. In this module, there are several classes that represent dates, time, and even timezones. One of the most flexible classes is `datetime.datetime` which represents the date and time together.

Here, we'll learn to perform datetime parsing and formatting using this class. Two operations that we'll consider are converting a string into a `datetime` and formatting a `datetime` object into a string. However, before going into details, we need to look closely at different formats.

§1. Formats

In order to parse or format dates, we need to specify the format. Take the date 06/04/2020, for example. Is this June 4 or April 6? Your answer will depend on where you're from. When we work with dates in Python, we don't want to play the guessing game. To avoid confusion, we just specify what format we want to work with.

The format codes for working with dates make use of **the % operator** taken from the C programming language. With its help, you can include all kinds of things: days of the week, timezone names, microseconds, but we'll focus on the main ones. The rest you can find in the official documentation.

Code	Meaning
%d	Day of the month — 01, 02, ..., 31
%m	Month as a number — 01, 02, ..., 12
%y	Year without century — 00, 01, ..., 99
%Y	Year with century — 0001, ..., 2020, ..., 9999
%H	Hour (24 hours) — 00, 01, ..., 23
%I	Hour (12 hours) — 01, 02, ..., 12
%M	Minutes — 00, 01, ..., 59
%S	Seconds — 00, 01, ..., 59
%B	Month as the area's full name — January, February, ..., December

Format codes

Formats indicate from where to extract the desired information. Going back to our example, if we want to process the date 06/04/2020 as June 4, then we need to use the format `"%m/%d/%Y"`. This is our way of saying that we have the sequence of the month, day, and year separated by slashes. If we choose April 6, then the format is `"%d/%m/%Y"`. Note that we give very explicit instructions in the format strings: all delimiters, spaces, and punctuation should be in place, otherwise the date will not be parsed or we will not get the desired formatted string.

> Pay attention to zeros in the formats! They are necessary for these particular formats that we've provided. If you want to discard zeros in

Current topic:

Datetime parsing and formatting

Topic depends on:

String formatting Stage 1

Datetime module Stage 2

Table of contents:

↑ Datetime parsing and formatting

§1. Formats

§2. Datetime parsing

§3. Datetime formatting

§4. Summary

Feedback & Comments

some fields, there are other formats you can use, check the documentation for that.

§2. Datetime parsing

Now that we know about formats, let's get into action! The first order of business is **datetime parsing**, that is, creating a `datetime` object from a string that represents some point in time in a particular format.

Datetime parsing is done with the `datetime.strptime()` method. This particular method name can be seen in many languages, not just Python. The method takes two arguments, `date_string` and `format`, and returns a `datetime` object that corresponds to this string.

When we use this method, we can omit the leading zero in some formats: `%d`, `%m`, `%H`, `%I`, `%M` and `%S`. Meaning that both 01 and 1 will be parsed correctly (if they fall in the range supported by the format).

Let's go back to our ambiguous date again. We are going to create a string with the date, add some time information to make things more interesting, and parse it into different `datetime` objects. The full date we've chosen is 06/04/2020 12:30.

```
from datetime import datetime


date_string = "06/04/2020 12:30"

dt1 = datetime.strptime(date_string, "%m/%d/%Y %H:%M")
print(dt1)
# 2020-06-04 12:30:00


dt2 = datetime.strptime(date_string, "%d/%m/%Y %H:%M")

print(dt2)

# 2020-04-06 12:30:00



dt3 = datetime.strptime(date_string, "%d/%m/%Y %I:%M")

print(dt3)

# 2020-04-06 00:30:00
```

We got three different datetime objects from one string simply because we specified different formats. The difference in the first two objects is the one we've already discussed: month first or day first. For both these dates, we've specified a 24-hour clock time and got 12:30 in the afternoon.

In the third case, we've got a different time! This is because in the format string we chose a 12-hour clock time where midnight is 12:00 AM and midday is 12:00 PM. We didn't specify the time period so by default the time got parsed as midnight. There is, of course, a format code for AM and PM that you can find in the documentation.

Again, you can see above that we've detailed the format strings and wrote down exactly how our strings are structured. If we're not careful and write an incorrect format or choose a wrong code, we'll get a `ValueError`:

```
datetime.strptime("06/04/2020", "%d/%m/%Y %I/%M")  # no time data in the original string
# ValueError: time data '06/04/2020' does not match format '%d/%m/%Y %I/%M'

datetime.strptime("06/04/2020 00:30", "%d/%m/%Y %I/%M")  # wrong time format
# ValueError: time data '06/04/2020 00:30' does not match format '%d/%m/%Y %I/%M'
```

§3. Datetime formatting

Now that we know how to parse strings into `datetime` objects, let's do the opposite and format `datetime` objects into strings. This can be done with the help of the `datetime.strftime()` method. To distinguish the two methods, you can think of this one as "**str**ing **f**rom **time**".

The only argument needed for this method is, as you might've guessed, the format. Let's take June 4, 2020, at 12:30 and format it in different ways.

```
dt4 = datetime(2020, 6, 4, 12, 30)

date_string1 = dt4.strftime("%B %d, %Y at %H:%M")
print(date_string1)
# June 04, 2020 at 12:30

date_string2 = dt4.strftime("%d.%m.%y")
print(date_string2)
# 04.06.20



date_string3 = dt4.strftime("%Y-%m-%d-%I:%m")

print(date_string3)

# 2020-06-04-12:06



date_string4 = dt4.strftime("The event will take place on %B %d.")

print(date_string4)

# The event will take place on June 04.
```

Here we've got three different strings from the object. You can see that our formats don't have to be comprehensive. We can include whatever data from the object that we need.

> Note also that to call the `strftime()` method, we need an instance of the `datetime` class. `strptime()` is different because it creates a `datetime` object, so it is called directly from the class.

§4. Summary

In this topic, we've seen how datetime parsing and formatting can be done in Python, specifically in the `datetime` class from the `datetime` module.

Parsing and formatting is done with `strptime` and `strftime` methods respectively. For both these methods, we need to specify the desired format using the standard C notation.

`strptime` takes two arguments, a string, and a format, and returns a `datetime` object corresponding to these string and format. `strftime` formats a `datetime` object into a string, taking only a format as an argument.

Report a typo

85 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

Comments (7) Hints (0) Useful links (1) Show discussion