The datetime Module

WE'LL COVER THE FOLLOWING

We will be learning about the following classes from the **datetime** module:

- datetime.date
- datetime.timedelta
- datetime.datetime

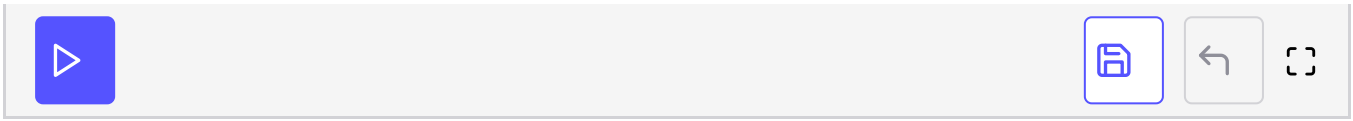
These will cover the majority of instances where you'll need to use date and datetime object in Python. There is also a **tzinfo** class for working with time zones that we won't be covering. Feel free to take a look at the Python documentation for more information on that class.

datetime.date

Python can represent dates several different ways. We're going to look at the **datetime.date** format first as it happens to be one of the simpler date objects.

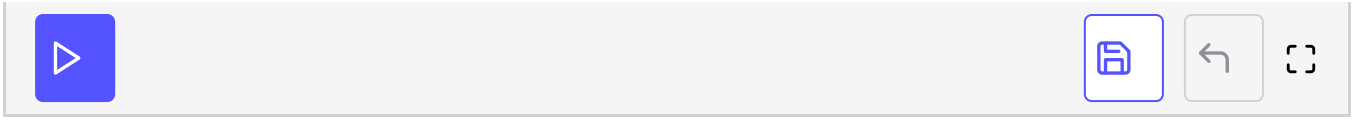
```
import datetime

datetime.date(2012, 13, 14)
# Traceback (most recent call last):
#  File "/usercode/__ed_file.py", line 3, in <module>
#    datetime.date(2012, 13, 14)
# ValueError: month must be in 1..12
```

```
import datetime

print(datetime.date(2012, 12, 14))
# 2012-12-14
```



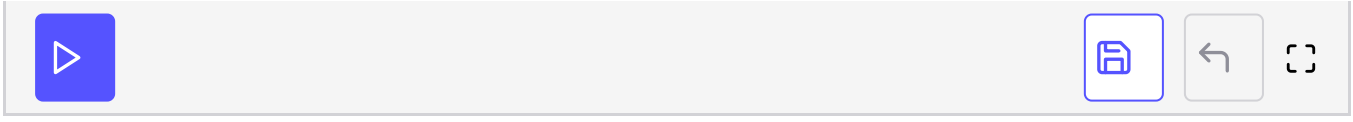
This code shows how to create a simple date object. The date class accepts three arguments: the year, the month and the day. If you pass it an invalid value, you will see a **ValueError**, like the one above. Otherwise you will see a **datetime.date** object returned. Let's take a look at another example:

```
import datetime
d = datetime.date(2012, 12, 14)
print(d.year)
#2012

print(d.day)
#14

print(d.month)
#12
```



Here we assign the date object to the variable **d**. Now we can access the various date components by name, such as **d.year** or **d.month**. Now let's find out what day it is:

```
import datetime

datetime.date.today()
print(datetime.date(2014, 3, 5))
#2014-03-05
```

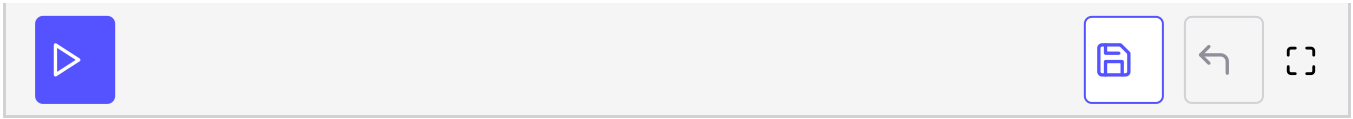


This can be helpful whenever you need to record what day it is. Or perhaps you need to do a date-based calculation based on today. It's a handy little convenience method though.

datetime.datetime

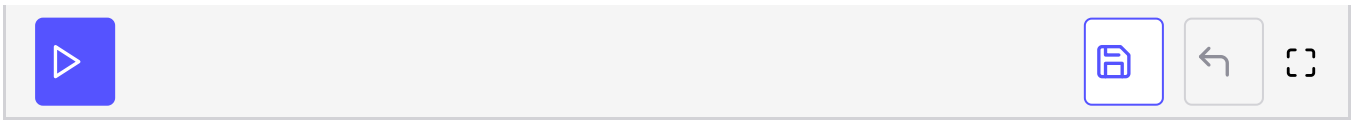
A **datetime.datetime** object contains all the information from a datetime.date plus a datetime.time object. Let's create a couple of examples so we can better understand the difference between this object and the datetime.date object.

```
import datetime

print(datetime.datetime(2014, 3, 5))
# 2014-03-05 00:00:00 datetime.datetime(2014, 3, 5, 0, 0)

print(datetime.datetime(2014, 3, 5, 12, 30, 10))
# 2014-03-05 12:30:10 datetime.datetime(2014, 3, 5, 12, 30, 10)

d = datetime.datetime(2014, 3, 5, 12, 30, 10)
print(d.year)
#2014

print(d.second)
#10

print(d.hour)
#12
```



Here we can see that **datetime.datetime** accepts several additional arguments: year, month, day, hour, minute and second. It also allows you to specify microsecond and timezone information too. When you work with databases, you will find yourself using these types of objects a lot. Most of the time, you will need to convert from the Python date or datetime format to the SQL datetime or timestamp format. You can find out what today is with datetime.datetime using two different methods:

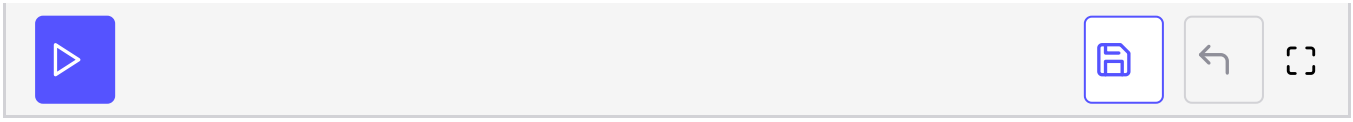
```
import datetime

print(datetime.datetime.today())
#2016-12-30 09:11:52.90927

print(datetime.datetime.now())
#2016-12-30 09:11:52.909304
```



The datetime module has another method that you should be aware of called

strftime. This method allows the developer to create a string that represents the time in a more human readable format. There's an entire table of formatting options that you should go read in the Python documentation, section 8.1.7. We're going to look at a couple of examples to show you the power of this method:

```
import datetime

print(datetime.datetime.today().strftime("%Y%m%d"))
#20161230

today = datetime.datetime.today()
print(today.strftime("%m/%d/%Y"))
#12/30/2016

print(today.strftime("%Y-%m-%d-%H.%M.%S"))
#2016-12-30-09.13.58
```



The first example is kind of a hack. It shows how to convert today's datetime object into a string that follows the YYYYMMDD (year, month, day) format. The second example is better. Here we assign today's datetime object to a variable called **today** and then try out two different string formatting operations. The first one adds forward slashes between the datetime elements and also rearranges it so that it becomes month, day, year. The last example creates a timestamp of sorts that follows a fairly typical format: YYYY-MM-DD.HH.MM.SS. If you want to go to a two-digit year, you can swap out the **%Y** for **%y**.

datetime.timedelta

The **datetime.timedelta** object represents a time duration. In other words, it is the difference between two dates or times. Let's take a look at a simple example:

```
import datetime

now = datetime.datetime.now()
print(now)
#2016-12-30 09:24:02.233647

then = datetime.datetime(2014, 2, 26)
delta = now - then
print(type(delta))
```

```
#<type 'datetime.timedelta'>

print(delta.days)
#1038

print(delta.seconds)
#33842
```

We create two datetime objects here. One for today and one for a week ago. Then we take the difference between them. This returns a timedelta object which we can then use to find out the number of days or seconds between the two dates. If you need to know the number of hours or minutes between the two, you'll have to use some math to figure it out. Here's one way to do it:

```
import datetime

now = datetime.datetime.now()
then = datetime.datetime(2014, 2, 26)
delta = now - then

seconds = delta.total_seconds()
hours = seconds // 3600
print(hours)
#24921.0

minutes = (seconds % 3600) // 60
print(minutes)
#34.0
```

What this tells us is that there are 186 hours and 13 minutes in a week. Note that we are using a double-forward slash as our division operator. This is known as **floor division**.

Now we're ready to move on and learn a bit about the **time** module!