**Daily XP450**

**What do you mean I don't have any class? Namedtuple**

**50 XP**

**1. namedtuple**

Often times when working with data, you will use a dictionary just so you can use key names to make reading the code and accessing the data easier to understand.

**2. What is a namedtuple?**

Python has another container called a namedtuple which is a tuple, but has names for each position of the tuple. This works well when you don't need the nested structure of a dictionary or desire each item to look identical, and don't want to add the overhead of a Pandas dataframe.

**3. Creating a namedtuple**

You create a namedtuple by passing a name for the tuple type and a list of field names. Let's begin by importing namedtuple from the collections module. Next we'll define our namedtuple. It's common practice to use Pascalcase (capitalizing each word) when naming namedtuples so hear I've used Eatery with a capital E for both the tuple name and the variable we stored the namedtuple as. Then we provide a list of fields we want on the nametuple. Now we can create our Eatery named tuples. I'm going to change our NYC Park eateries data into a list of namedtuples. I create an empty list then iterate over my nyc\_eateries list creating an instance of my Eatery namedtuple by passing in the data from the loop as the arguments to my namedtuple.

**4. Print the first element**

Finally, let's print the first Eatery in the list. Now that we've got a list of named tuples let's see how we can use them.

**5. Leveraging namedtuples**

One of the great things about named tuples is that they can make code clearer because each field is available an as attribute. An attribute is basically a named field or data storage location. We can also depend on every instance of a namedtuple to have all the fields, although some might be empty or None in Python terms. This means we can always have safe access to a field without the need for a get method like a dictionary. Here I'm going to use the list of tuples we created in the prior slide, and print the name, park\_id, and location for the first three entries in the list. Now it's your turn to practice.

**6. Let's practice!**

# Creating namedtuples for storing data

Often times when working with data, you will use a dictionary just so you can use key names to make reading the code and accessing the data easier to understand. Python has another container called a namedtuple that is a tuple, but has names for each position of the tuple. You create one by passing a name for the tuple type and a list of field names.

For example, Cookie = namedtuple("Cookie", ['name', 'quantity']) will create a container, and you can create new ones of the type using Cookie('chocolate chip', 1) where you can access the name using the name attribute, and then get the quantity using the quantity attribute.

In this exercise, you're going to restructure the transit data you've been working with into namedtuples for more descriptive code.

##### Instructions

**100 XP**

* Import namedtuple from collections.
* Create a namedtuple called DateDetails with a type name of DateDetails and fields of 'date', 'stop', and 'riders'.
* Create a list called labeled\_entries.
* Iterate over the entries list, unpacking it into date, stop, and riders.
* Create a new DateDetails namedtuple instance for each entry and append it to labeled\_entries.
* Print the first 5 items in labeled\_entries. This has been done for you, so hit 'Submit Answer' to see the result!

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for \_\_\_\_, \_\_\_\_, \_\_\_\_ in \_\_\_\_:

    # Append a new DateDetails namedtuple instance for each entry to labeled\_entries

    \_\_\_\_

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

# Print the first key in ridership\_date print(list(ridership\_date.keys())[0]) # Pop the first item from ridership\_date and print it print(ridership\_date.popitem()) # Print the last key in ridership\_date print(list(ridership\_date.keys())[-1]) # Pop the last item from ridership\_date and print it print(ridership\_date.popitem(last=False))

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

    # Append a new DateDetails namedtuple instance for each entry to labeled\_entries

    DateDetails1 = DateDetails(date, stop, riders)

    labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

* IPython Shell
* Slides

# Print the first key in ridership\_date

print(list(ridership\_date.keys())[0])

# Pop the first item from ridership\_date and print it

print(ridership\_date.popitem())

# Print the last key in ridership\_date

print(list(ridership\_date.keys())[-1])

# Pop the last item from ridership\_date and print it

print(ridership\_date.popitem(last=False))

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails.append(labeled\_entries)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 13, in <module>

DateDetails.append(labeled\_entries)

AttributeError: type object 'DateDetails' has no attribute 'append'

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails = 'DateDetails'(datedetails['date'],datedetails['stop'],datedetails['riders'] )

entries.append(labeled\_entries)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 13, in <module>

DateDetails = 'DateDetails'(datedetails['date'],datedetails['stop'],datedetails['riders'] )

NameError: name 'datedetails' is not defined

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

labeled\_entries = 'DateDetails'(datedetails['date'],datedetails['stop'],datedetails['riders'] )

entries.append(labeled\_entries)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 13, in <module>

labeled\_entries = 'DateDetails'(datedetails['date'],datedetails['stop'],datedetails['riders'] )

NameError: name 'datedetails' is not defined

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

labeled\_entries = 'DateDetails'([datedetails['date'],datedetails['stop'],datedetails['riders']] )

entries.append(labeled\_entries)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 13, in <module>

labeled\_entries = 'DateDetails'([datedetails['date'],datedetails['stop'],datedetails['riders']] )

NameError: name 'datedetails' is not defined

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

labeled\_entries = NewDateDetails

entries.append(labeled\_entries)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 13, in <module>

labeled\_entries = NewDateDetails

NameError: name 'NewDateDetails' is not defined

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

labeled\_entries.append(DateDetails)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

[<class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>]

<script.py> output:

[<class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>]

<script.py> output:

[<class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>, <class '\_\_main\_\_.DateDetails'>]

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

labeled\_entries.append(DateDetails['date'], DateDetails['stop'], DateDaetails['riders'])

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 13, in <module>

labeled\_entries.append(DateDetails['date'], DateDetails['stop'], DateDaetails['riders'])

NameError: name 'DateDaetails' is not defined

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails()

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 13, in <module>

DateDetails1 = DateDetails()

TypeError: <lambda>() missing 3 required positional arguments: 'date', 'stop', and 'riders'

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails('date', 'stop', 'riders')

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

[DateDetails(date='date', stop='stop', riders='riders'), DateDetails(date='date', stop='stop', riders='riders'), DateDetails(date='date', stop='stop', riders='riders'), DateDetails(date='date', stop='stop', riders='riders'), DateDetails(date='date', stop='stop', riders='riders')]

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails(['date', 'stop', 'riders'])

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 13, in <module>

DateDetails1 = DateDetails(['date', 'stop', 'riders'])

TypeError: <lambda>() missing 2 required positional arguments: 'stop' and 'riders'

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails(['date'], ['stop'], ['riders'])

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

[DateDetails(date=['date'], stop=['stop'], riders=['riders']), DateDetails(date=['date'], stop=['stop'], riders=['riders']), DateDetails(date=['date'], stop=['stop'], riders=['riders']), DateDetails(date=['date'], stop=['stop'], riders=['riders']), DateDetails(date=['date'], stop=['stop'], riders=['riders'])]

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails([], [], [])

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

[DateDetails(date=[], stop=[], riders=[]), DateDetails(date=[], stop=[], riders=[]), DateDetails(date=[], stop=[], riders=[]), DateDetails(date=[], stop=[], riders=[]), DateDetails(date=[], stop=[], riders=[])]

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails([labeled\_entries], [], [])

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

[DateDetails(date=[[DateDetails(date=[...], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[])]], stop=[], riders=[]), DateDetails(date=[[DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[...], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[])]], stop=[], riders=[]), DateDetails(date=[[DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[...], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[])]], stop=[], riders=[]), DateDetails(date=[[DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[...], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[])]], stop=[], riders=[]), DateDetails(date=[[DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[...], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[]), DateDetails(date=[[...]], stop=[], riders=[])]], stop=[], riders=[])]

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails([], [], [])

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

[DateDetails(date=[], stop=[], riders=[]), DateDetails(date=[], stop=[], riders=[]), DateDetails(date=[], stop=[], riders=[]), DateDetails(date=[], stop=[], riders=[]), DateDetails(date=[], stop=[], riders=[])]

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails([date], [stop], [riders])

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

[DateDetails(date=['11/17/2016'], stop=['Dempster-Skokie'], riders=['2048']), DateDetails(date=['02/19/2016'], stop=['Adams/Wabash'], riders=['8447']), DateDetails(date=['03/25/2015'], stop=['State/Lake'], riders=['10325']), DateDetails(date=['05/02/2015'], stop=['Ashland-Lake'], riders=['1424']), DateDetails(date=['07/11/2015'], stop=['Chicago/Franklin'], riders=['2883'])]

<script.py> output:

[DateDetails(date=['11/17/2016'], stop=['Dempster-Skokie'], riders=['2048']), DateDetails(date=['02/19/2016'], stop=['Adams/Wabash'], riders=['8447']), DateDetails(date=['03/25/2015'], stop=['State/Lake'], riders=['10325']), DateDetails(date=['05/02/2015'], stop=['Ashland-Lake'], riders=['1424']), DateDetails(date=['07/11/2015'], stop=['Chicago/Franklin'], riders=['2883'])]

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails(date, stop, riders)

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

[DateDetails(date='11/17/2016', stop='Dempster-Skokie', riders='2048'), DateDetails(date='02/19/2016', stop='Adams/Wabash', riders='8447'), DateDetails(date='03/25/2015', stop='State/Lake', riders='10325'), DateDetails(date='05/02/2015', stop='Ashland-Lake', riders='1424'), DateDetails(date='07/11/2015', stop='Chicago/Franklin', riders='2883')]

<script.py> output:

**Daily XP600**

##### Exercise

##### Exercise

# Leveraging attributes on namedtuples

Once you have a namedtuple, you can write more expressive code that is easier to understand. Remember, you can access the elements in the tuple by their name as an attribute. For example, you can access the date of the namedtuples in the previous exercise using the .date attribute.

Here, you'll use the tuples you made in the previous exercise to see how this works.

##### Instructions

**100 XP**

* Iterate over the first twenty items in the labeled\_entries list:
  + Print each item's stop.
  + Print each item's date.
  + Print each item's riders.
* # Iterate over the first twenty items in labeled\_entries
* for \_\_\_\_ in \_\_\_\_:
* # Print each item's stop
* \_\_\_\_
* # Print each item's date
* \_\_\_\_
* # Print each item's riders
* \_\_\_\_
* script.py

1

2

3

4

5

6

7

8

9

10



* IPython Shell
* Slides

# Import namedtuple from collections

from collections import namedtuple

# Create the namedtuple: DateDetails

DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled\_entries

labeled\_entries = []

# Iterate over the entries list

for date, stop, riders in entries:

# Append a new DateDetails namedtuple instance for each entry to labeled\_entries

DateDetails1 = DateDetails(date, stop, riders)

labeled\_entries.append(DateDetails1)

# Print the first 5 items in labeled\_entries

print(labeled\_entries[:5])

# Iterate over the first twenty items in labeled\_entries

for item in labeled\_entries[:20]:

    # Print each item's stop

    print(item.stop)

    # Print each item's date

    print(item.date)

    # Print each item's riders

    print(item.riders)

Iterate over the first twenty items in labeled\_entries

for item in labeled\_entries[:20]:

# Print each item's stop

print(item.stop)

# Print each item's date

print(item.date)

# Print each item's riders

print(item.riders)

Austin-Forest Park

01/01/2015

587

Austin-Forest Park

01/02/2015

1386

Austin-Forest Park

01/03/2015

785

Austin-Forest Park

01/04/2015

625

Austin-Forest Park

01/05/2015

1752

Austin-Forest Park

01/06/2015

1777

Austin-Forest Park

01/07/2015

1269

Austin-Forest Park

01/08/2015

1435

Austin-Forest Park

01/09/2015

1631

Austin-Forest Park

01/10/2015

771

Austin-Forest Park

01/11/2015

588

Austin-Forest Park

01/12/2015

2065

Austin-Forest Park

01/13/2015

2108

Austin-Forest Park

01/14/2015

2012

Austin-Forest Park

01/15/2015

2069

Austin-Forest Park

01/16/2015

2003

Austin-Forest Park

01/17/2015

953

Austin-Forest Park

01/18/2015

706

Austin-Forest Park

01/19/2015

1216

Austin-Forest Park

01/20/2015

2115

<script.py> output:

Austin-Forest Park

01/01/2015

587

Austin-Forest Park

01/02/2015

1386

Austin-Forest Park

01/03/2015

785

Austin-Forest Park

01/04/2015

625

Austin-Forest Park

01/05/2015

1752

Austin-Forest Park

01/06/2015

1777

Austin-Forest Park

01/07/2015

1269

Austin-Forest Park

01/08/2015

1435

Austin-Forest Park

01/09/2015

1631

Austin-Forest Park

01/10/2015

771

Austin-Forest Park

01/11/2015

588

Austin-Forest Park

01/12/2015

2065

Austin-Forest Park

01/13/2015

2108

Austin-Forest Park

01/14/2015

2012

Austin-Forest Park

01/15/2015

2069

Austin-Forest Park

01/16/2015

2003

Austin-Forest Park

01/17/2015

953

Austin-Forest Park

01/18/2015

706

Austin-Forest Park

01/19/2015

1216

Austin-Forest Park

01/20/2015

2115

In [1]:

**Daily XP700**

**There and Back Again a DateTime Journey**

**50 XP**

**1. There and Back Again a DateTime Journey**

Dealing with dates and times is often considered to be very confusing with all the considerations due to the unique ways in which time flows. Leap Years, Different length months, different distribution of weekdays/weekends, and the dreaded timezones are just a few of the things we must consider. However, with careful reasoning, you'll soon be working with datetime data with relative ease. Only practice and experience can make you fluent in datetime issues, so let's start on this learning journey. I'll be using data from the NYC parking violations dataset.

**2. From string to datetime**

When working with datetimes in Python, we use the datetime module from the standard library. There is a datetime type inside of the datetime module. In addition to letting us manually create datetime objects, we can also parse existing strings into datetime objects. This is an exceptionally common task. So let's dive into an example. We being by importing the datetime type from the datetime module. Next, I'm printing the string that we are going to parse to create a datetime object.

**3. Parsing strings into datetimes**

Next I use the strptime method to parse the parking\_violations\_date using a format string for MM/DD/YYYY (more on this in just one sec). Finally, I print the datetime object so we can see the results. Okay, so about those format strings...

**4. Time Format Strings**

Time format strings are common across many different programming language, and originated in C. Here are the few that we just used with the strptime method. You use the directives and any special characters or spaces you need to match the datestring you are trying to parse. There is a complete list of them in the Python Documentation, and I linked in the exercises for you to reference. In addition to using these format strings for converting from a string to a datetime,

**5. Datetime to String**

we can also use them to go from a datetime to a string object with the strftime() method. For example, we can use the same format string to output the datetime we created on the prior slide. We can also output the string as an ISO standard datetime string, one of the most common ways to express times when dealing with writing to or reading from files or applications. Let's use it on our datetime object, we've been working with. Time for you to practice.

**6. Let's practice!**

**Daily XP750**

##### Exercise

##### Exercise

# Strings to DateTimes

Time to begin your DateTime journey! You'll start by using the .strptime() method from the datetime object as shown in the video, passing it both the string and the format. A full list of the format string components is available in the Python [documentation](https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior).

You'll be using the datetime column from the Chicago Transit Authority data, which is available as dates\_list. Feel free to explore it in the IPython Shell: You'll see that it has the format of Month, Day, Year.

##### Instructions

**100 XP**

* Import the datetime object from datetime.
* Iterate over the dates\_list, using date\_str as your iterator variable.
* Convert each date\_str into a datetime object called date\_dt using the datetime.strptime() function, with '%m/%d/%Y' as your format.
* Print each date\_dt.
* # Import the datetime object from datetime
* \_\_\_\_
* # Iterate over the dates\_list
* for \_\_\_\_ in \_\_\_\_:
* # Convert each date to a datetime object: date\_dt
* date\_dt = \_\_\_\_
* # Print each date\_dt
* print(\_\_\_\_)

# Iterate over the first twenty items in labeled\_entries for item in labeled\_entries[:20]: # Print each item's stop print(item.stop) # Print each item's date print(item.date) # Print each item's riders print(item.riders)

**Daily XP850**

##### Exercise

##### Exercise

# Strings to DateTimes

Time to begin your DateTime journey! You'll start by using the .strptime() method from the datetime object as shown in the video, passing it both the string and the format. A full list of the format string components is available in the Python [documentation](https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior).

You'll be using the datetime column from the Chicago Transit Authority data, which is available as dates\_list. Feel free to explore it in the IPython Shell: You'll see that it has the format of Month, Day, Year.

##### Instructions

**100 XP**

* Import the datetime object from datetime.
* Iterate over the dates\_list, using date\_str as your iterator variable.
* Convert each date\_str into a datetime object called date\_dt using the datetime.strptime() function, with '%m/%d/%Y' as your format.
* Print each date\_dt.

+100 XP

Great work! You now know how to convert Strings to DateTimes. In the next exercise, you'll learn how to do the opposite!

Press enter to

* script.py

1

2

3

4

5

6

7

8



* IPython Shell
* Slides

# Iterate over the first twenty items in labeled\_entries

for item in labeled\_entries[:20]:

# Print each item's stop

print(item.stop)

# Print each item's date

print(item.date)

# Print each item's riders

print(item.riders)

In [1]:

print(dates\_list)

['02/19/2001', '04/10/2001', '05/30/2001', '07/19/2001', '09/07/2001', '10/27/2001', '12/16/2001', '02/04/2002', '03/26/2002', '05/15/2002', '07/04/2002', '08/23/2002', '10/12/2002', '12/01/2002', '01/20/2003', '03/11/2003', '04/30/2003', '06/19/2003', '08/08/2003', '09/27/2003', '11/16/2003', '01/05/2004', '02/24/2004', '04/14/2004', '06/03/2004', '07/23/2004', '09/11/2004', '10/31/2004', '12/20/2004', '02/08/2005', '03/30/2005', '05/19/2005', '07/08/2005', '08/27/2005', '10/16/2005', '12/05/2005', '01/24/2006', '03/15/2006', '05/04/2006', '06/23/2006', '08/12/2006', '10/01/2006', '11/20/2006', '01/09/2007', '02/28/2007', '04/19/2007', '06/08/2007', '07/28/2007', '09/16/2007', '11/05/2007', '12/25/2007', '02/13/2008', '04/03/2008', '05/23/2008', '07/12/2008', '08/31/2008', '10/20/2008', '12/09/2008', '01/28/2009', '03/19/2009', '05/08/2009', '06/27/2009', '08/16/2009', '10/05/2009', '11/24/2009', '01/13/2010', '03/04/2010', '04/23/2010', '06/12/2010', '08/01/2010', '09/20/2010', '11/09/2010', '12/29/2010', '02/17/2011', '04/08/2011', '05/28/2011', '07/17/2011', '09/05/2011', '10/24/2011', '11/12/2011', '01/01/2012', '02/20/2012', '04/10/2012', '05/30/2012', '07/19/2012', '09/07/2012', '10/27/2012', '12/16/2012', '02/04/2013', '03/26/2013', '05/15/2013', '07/04/2013', '08/23/2013', '10/12/2013', '12/01/2013', '01/20/2014', '03/11/2014', '04/30/2014', '06/19/2014', '08/08/2014', '09/27/2014', '11/16/2014', '07/05/2014', '01/24/2015', '03/15/2015', '05/04/2015', '06/23/2015', '08/12/2015', '10/01/2015', '11/20/2015', '01/09/2016', '02/28/2016', '04/18/2016', '06/07/2016', '07/27/2016', '09/15/2016', '11/04/2016']

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = datetime.strptime(dates\_list,'%m/%d/%Y')

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = datetime.strptime(dates\_list,'%m/%d/%Y')

TypeError: strptime() argument 1 must be str, not list

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = datetime.strptime(dates\_str,'%m/%d/%Y')

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = datetime.strptime(dates\_str,'%m/%d/%Y')

NameError: name 'dates\_str' is not defined

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = datetime.strptime(dates\_str,'%m/%d/%Y')

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = datetime.strptime(dates\_str,'%m/%d/%Y')

NameError: name 'dates\_str' is not defined

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = datetime.strptime(dates\_list,'%m/%d/%Y')

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = datetime.strptime(dates\_list,'%m/%d/%Y')

TypeError: strptime() argument 1 must be str, not list

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = datetime.strptime(dates\_str,'%m/%d/%Y')

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = datetime.strptime(dates\_str,'%m/%d/%Y')

NameError: name 'dates\_str' is not defined

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = datetime.strptime('%m/%d/%Y')

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = datetime.strptime('%m/%d/%Y')

TypeError: strptime() takes exactly 2 arguments (1 given)

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = (list(datetime.strptime('%m/%d/%Y')))

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = (list(datetime.strptime('%m/%d/%Y')))

TypeError: strptime() takes exactly 2 arguments (1 given)

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = (list(datetime.strptime(dates\_list'%m/%d/%Y')))

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 2944, in \_run\_cell

return runner(coro)

File "/var/lib/python/site-packages/python3.9/IPython/core/async\_helpers.py", line 68, in \_pseudo\_sync\_runner

coro.send(None)

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 3150, in run\_cell\_async

self.showsyntaxerror()

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 2114, in showsyntaxerror

self.\_showtraceback(etype, value, stb)

File "/var/lib/python/site-packages/python3.9/pythonbackend/shell\_utils.py", line 72, in exceptionCatcher

raise exception

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 3140, in run\_cell\_async

code\_ast = compiler.ast\_parse(cell, filename=cell\_name)

File "/var/lib/python/site-packages/python3.9/IPython/core/compilerop.py", line 101, in ast\_parse

return compile(source, filename, symbol, self.flags | PyCF\_ONLY\_AST, 1)

File "<stdin>", line 6

date\_dt = (list(datetime.strptime(dates\_list'%m/%d/%Y')))

^

SyntaxError: invalid syntax

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = (list(datetime.strptime(dates\_list, '%m/%d/%Y')))

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = (list(datetime.strptime(dates\_list, '%m/%d/%Y')))

TypeError: strptime() argument 1 must be str, not list

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = datetime.strptime(list(dates\_list, '%m/%d/%Y'))

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = datetime.strptime(list(dates\_list, '%m/%d/%Y'))

TypeError: list expected at most 1 argument, got 2

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = datetime.strptime(dates\_list, '%m/%d/%Y')

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = datetime.strptime(dates\_list, '%m/%d/%Y')

TypeError: strptime() argument 1 must be str, not list

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

# Convert each date to a datetime object: date\_dt

date\_dt = datetime.strptime(dates\_str, '%m/%d/%Y')

# Print each date\_dt

print(date\_dt)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 6, in <module>

date\_dt = datetime.strptime(dates\_str, '%m/%d/%Y')

NameError: name 'dates\_str' is not defined

# Import the datetime object from datetime

from datetime import datetime

# Iterate over the dates\_list

for date\_str in dates\_list:

    # Convert each date to a datetime object: date\_dt

    date\_dt = datetime.strptime(date\_str, '%m/%d/%Y')

    # Print each date\_dt

    print(date\_dt)

**Daily XP850**

##### Exercise

##### Exercise

# Converting to a String

Converting from a datetime object to a string is done with the .strftime() method on a instance of the datetime object. You pass a format string just like the ones used in the prior exercise.

There is also a widely used string output standard called ISO-8601. It has a shortcut method named .isoformat(). I encourage you to use it anytime you write out to a file.

All the datetimes you created for the transit data in the prior exercise are saved in the datetimes\_list.

##### Instructions

**100 XP**

* Loop over the first 10 items of the datetimes\_list, using item as your iterator variable.
  + Print out the item as a string in the format of 'MM/DD/YYYY'. For this, the format string is '%m/%d/%Y'.
  + Print out the item as an ISO standard string.

# Loop over the first 10 items of the datetimes\_list

for item in datetimes\_list[:10]:

    # Print out the record as a string in the format of 'MM/DD/YYYY'

    print(\_\_\_\_)

    # Print out the record as an ISO standard string

    print(\_\_\_\_)

# Import the datetime object from datetime from datetime import datetime # Iterate over the dates\_list for date\_str in dates\_list: # Convert each date to a datetime object: date\_dt date\_dt = datetime.strptime(date\_str, '%m/%d/%Y') # Print each date\_dt print(date\_dt)

# Loop over the first 10 items of the datetimes\_list

for item in datetimes\_list[:10]:

    # Print out the record as a string in the format of 'MM/DD/YYYY'

    print(item.strftime('%m/%d/%Y'))

    # Print out the record as an ISO standard string

    print(item.isoformat())

# Loop over the first 10 items of the datetimes\_list

for item in datetimes\_list[:10]:

# Print out the record as a string in the format of 'MM/DD/YYYY'

print(item.strftime('%m/%d/%Y'))

# Print out the record as an ISO standard string

print(item.isoformat())

02/19/2001

2001-02-19T00:00:00

04/10/2001

2001-04-10T00:00:00

05/30/2001

2001-05-30T00:00:00

07/19/2001

2001-07-19T00:00:00

09/07/2001

2001-09-07T00:00:00

10/27/2001

2001-10-27T00:00:00

12/16/2001

2001-12-16T00:00:00

02/04/2002

2002-02-04T00:00:00

03/26/2002

2002-03-26T00:00:00

05/15/2002

2002-05-15T00:00:00

**Daily XP950**

**Working with Datetime Components and current time**

**50 XP**

**1. Working with Datetime Components and current time**

Once we have a datetime object, we can work with it to get parts of the datetime like the month, year, or day. We can also get the current time and manipulate a timezone. Let's begin by working with the parts of a datetime object.

**2. Datetime Components**

All the parts of a datetime object are available as attributes, such as day, month, year, hour, minute, second etc. These are often used to group data by a particular time frame. Let's count the NYC parking violations for 2016 and group by day. We'll start by using a defaultdict of ints to count the records by day. Next, I'll loop over the packing violations to and parse the date which is found in the fifth element of our list. Next I increment the appropriate day based on the violation.

**3. Datetime Components - Results**

Finally I'm going to sort the days of the month and print the day of the month and our violation count for that day.

**4. What is the deal with now**

Often when working with datetime objects, you'll want to work on windows or ranges that start from the current date and time. We can do this using the datetime now functions. There is a dot now() method on the datetime object in the datetime module and a dot utcnow() method. The dot now() method returns the current local time on the machine on which it is run, and dot utcnow() does the same thing but returns the value in UTC timezone. We'll talk more about timezones in a second. Let's start by importing datetime type. Then we'll call the now method on it and it will return a new datetime representing the current time. Let's print it to see what we got.

**5. What is the deal with utcnow**

Next, let's call utcnow and get the current time in the UTC timezone. Finally, print it. The UTC timezone is only timezone with this special kind of method. Let's learn a bit more about timezones.

**6. Timezones**

Timezones can make life very interesting! By default, any datetime you make using the now methods are "naive" datetime objects, which means they are missing their timezone that is required to make an "aware" datetime object. You'll often get data where timezone is not supplied and you need to set it manually. In order to work effectively with other timezones, you can use the pytz module and use the timezone names from the Olsen database, the standard for timezone information. An "aware" datetime object has an dot astimezone() method that accepts a timezone object and returns a new datetime object in the desired timezone. If the tzinfo is not set for the datetime object it assumes the timezone of the computer you are working on. Let's see this in action.

**7. Timezones in action**

I'll begin by importing timezone from pytz. I've got the datetime of a violation, and parsed it into a naive datetime object. Next, I'm going to prepare the timezone objects I'm going to work with. Then, I create an object to present the Eastern timezone that NYC is in, and then the timezone for LA. Next, I use the replace method to replace the empty timezone on my record datetime and save it as my datetime. Now that I have an aware datetime instance, I can use the as timezone method to get the records time in LA.

**8. Timezones in action - results**

Finally, I print them both. This type of conversion is great for ensuring that data remains relevant to the viewer by presenting it in their native timezone. Your turn to work with these concepts.

**9. Let's practice!**

**Daily XP1000**

##### Exercise

##### Exercise

# Pieces of Time

When working with datetime objects, you'll often want to group them by some component of the datetime such as the month, year, day, etc. Each of these are available as attributes on an instance of a datetime object.

You're going to work with the summary of the CTA's daily ridership. It contains the following columns, in order: service\_date, day\_type, bus, rail\_boardings, and total\_rides. The modules defaultdict and datetime have already been imported for you.

##### Instructions

**100 XP**

* Create a defaultdict of an integer called monthly\_total\_rides.
* Loop over the list daily\_summaries, which contains the columns mentioned above in the assignment text.
  + Convert the service\_date (1st element of daily\_summary) to a datetime object called service\_datetime. Use '%m/%d/%Y' as your format string.
  + Use the month of the service\_datetime as the dict key and add the total\_rides (5th element of daily\_summary) to the current amount for the month. Be sure to convert this into an integer.
* Print monthly\_total\_rides.

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = \_\_\_\_

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

    # Convert the service\_date to a datetime object

    service\_datetime = \_\_\_\_

    # Add the total rides to the current amount for the month

    monthly\_total\_rides[\_\_\_\_] += int(\_\_\_\_)

# Print monthly\_total\_rides

print(monthly\_total\_rides)

# Loop over the first 10 items of the datetimes\_list for item in datetimes\_list[:10]: # Print out the record as a string in the format of 'MM/DD/YYYY' print(item.strftime('%m/%d/%Y')) # Print out the record as an ISO standard string print(item.isoformat())

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

    # Convert the service\_date to a datetime object

    service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

    # Add the total rides to the current amount for the month

    monthly\_total\_rides[service\_datetime.month] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

* IPython Shell
* Slides

# Loop over the first 10 items of the datetimes\_list

for item in datetimes\_list[:10]:

# Print out the record as a string in the format of 'MM/DD/YYYY'

print(item.strftime('%m/%d/%Y'))

# Print out the record as an ISO standard string

print(item.isoformat())

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(service\_date, '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime[month]+= int(total\_rides)

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 2944, in \_run\_cell

return runner(coro)

File "/var/lib/python/site-packages/python3.9/IPython/core/async\_helpers.py", line 68, in \_pseudo\_sync\_runner

coro.send(None)

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 3150, in run\_cell\_async

self.showsyntaxerror()

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 2114, in showsyntaxerror

self.\_showtraceback(etype, value, stb)

File "/var/lib/python/site-packages/python3.9/pythonbackend/shell\_utils.py", line 72, in exceptionCatcher

raise exception

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 3140, in run\_cell\_async

code\_ast = compiler.ast\_parse(cell, filename=cell\_name)

File "/var/lib/python/site-packages/python3.9/IPython/core/compilerop.py", line 101, in ast\_parse

return compile(source, filename, symbol, self.flags | PyCF\_ONLY\_AST, 1)

File "<stdin>", line 10

monthly\_total\_rides[service\_datetime[month]+= int(total\_rides)

^

SyntaxError: invalid syntax

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(service\_date, '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime[month]] += int(total\_rides)

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 7, in <module>

service\_datetime = datetime.strptime(service\_date, '%m/%d/%Y')

NameError: name 'service\_date' is not defined

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime('service\_date', '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime[month]] += int(total\_rides)

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 7, in <module>

service\_datetime = datetime.strptime('service\_date', '%m/%d/%Y')

File "<stdin>", line 568, in \_strptime\_datetime

tt, fraction, gmtoff\_fraction = \_strptime(data\_string, format)

File "<stdin>", line 349, in \_strptime

raise ValueError("time data %r does not match format %r" %

ValueError: time data 'service\_date' does not match format '%m/%d/%Y'

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime[month]] += int(total\_rides)

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[service\_datetime[month]] += int(total\_rides)

NameError: name 'month' is not defined

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime[month]] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[service\_datetime[month]] += int(daily\_summary[4])

NameError: name 'month' is not defined

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime['%m']] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[service\_datetime['%m']] += int(daily\_summary[4])

TypeError: 'datetime.datetime' object is not subscriptable

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[month]] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 2944, in \_run\_cell

return runner(coro)

File "/var/lib/python/site-packages/python3.9/IPython/core/async\_helpers.py", line 68, in \_pseudo\_sync\_runner

coro.send(None)

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 3150, in run\_cell\_async

self.showsyntaxerror()

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 2114, in showsyntaxerror

self.\_showtraceback(etype, value, stb)

File "/var/lib/python/site-packages/python3.9/pythonbackend/shell\_utils.py", line 72, in exceptionCatcher

raise exception

File "/var/lib/python/site-packages/python3.9/IPython/core/interactiveshell.py", line 3140, in run\_cell\_async

code\_ast = compiler.ast\_parse(cell, filename=cell\_name)

File "/var/lib/python/site-packages/python3.9/IPython/core/compilerop.py", line 101, in ast\_parse

return compile(source, filename, symbol, self.flags | PyCF\_ONLY\_AST, 1)

File "<stdin>", line 10

monthly\_total\_rides[month]] += int(daily\_summary[4])

^

SyntaxError: unmatched ']'

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[month] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[month] += int(daily\_summary[4])

NameError: name 'month' is not defined

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime(month)] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[service\_datetime(month)] += int(daily\_summary[4])

NameError: name 'month' is not defined

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime('month')] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[service\_datetime('month')] += int(daily\_summary[4])

TypeError: 'datetime.datetime' object is not callable

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime[month]] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[service\_datetime[month]] += int(daily\_summary[4])

NameError: name 'month' is not defined

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime['month']] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[service\_datetime['month']] += int(daily\_summary[4])

TypeError: 'datetime.datetime' object is not subscriptable

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service-date['month']] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[service-date['month']] += int(daily\_summary[4])

NameError: name 'service' is not defined

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_date['month']] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 10, in <module>

monthly\_total\_rides[service\_date['month']] += int(daily\_summary[4])

NameError: name 'service\_date' is not defined

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides['service\_date'['month']] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime.month] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

defaultdict(<class 'int'>, {1: 515062454, 2: 500276873, 3: 557894281, 4: 544878980, 5: 564403630, 6: 553707053, 7: 552970459, 8: 558434623, 9: 574770898, 10: 652199892, 11: 538491629, 12: 500560093})

<script.py> output:

defaultdict(<class 'int'>, {1: 515062454, 2: 500276873, 3: 557894281, 4: 544878980, 5: 564403630, 6: 553707053, 7: 552970459, 8: 558434623, 9: 574770898, 10: 652199892, 11: 538491629, 12: 500560093}

**Daily XP100**

##### Exercise

##### Exercise

# Creating DateTime Objects... Now

Often when working with datetime objects, you'll want to work on windows or ranges that start from the current date and time. You can do this using the datetime now functions. There is a .now() method on the datetime object in the datetime module and a .utcnow() method. The .now() method returns the current local time on the machine on which it is run, and .utcnow() does the same thing but returns the value in UTC time. You'll need to be very familiar with these methods.

No dataset is used in this exercise, but bear with us as you'll need to do this often to compare year/month-to-date etc.

##### Instructions

**100 XP**

* Import datetime from the datetime module.
* Store the local datetime as local\_dt and print it.
* Store the UTC datetime as utc\_dt and print it.

# Import datetime from the datetime module

\_\_\_\_

# Compute the local datetime: local\_dt

local\_dt = \_\_\_\_

# Print the local datetime

print(local\_dt)

# Compute the UTC datetime: utc\_dt

utc\_dt = \_\_\_\_

# Print the UTC datetime

print(utc\_dt)

# Create a defaultdict of an integer: monthly\_total\_rides monthly\_total\_rides = defaultdict(int) # Loop over the list daily\_summaries for daily\_summary in daily\_summaries: # Convert the service\_date to a datetime object service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y') # Add the total rides to the current amount for the month monthly\_total\_rides[service\_datetime.month] += int(daily\_summary[4]) # Print monthly\_total\_rides print(monthly\_total\_rides)

# Import datetime from the datetime module

from datetime import datetime

# Compute the local datetime: local\_dt

local\_dt = datetime.now()

# Print the local datetime

print(local\_dt)

# Compute the UTC datetime: utc\_dt

utc\_dt = datetime.utcnow()

# Print the UTC datetime

print(utc\_dt)



* IPython Shell
* Slides

# Create a defaultdict of an integer: monthly\_total\_rides

monthly\_total\_rides = defaultdict(int)

# Loop over the list daily\_summaries

for daily\_summary in daily\_summaries:

# Convert the service\_date to a datetime object

service\_datetime = datetime.strptime(daily\_summary[0], '%m/%d/%Y')

# Add the total rides to the current amount for the month

monthly\_total\_rides[service\_datetime.month] += int(daily\_summary[4])

# Print monthly\_total\_rides

print(monthly\_total\_rides)

# Import datetime from the datetime module

from datetime import datetime

# Compute the local datetime: local\_dt

local\_dt = datetime.now()

# Print the local datetime

print(local\_dt)

# Compute the UTC datetime: utc\_dt

utc\_dt = datetime.utcnow()

# Print the UTC datetime

print(utc\_dt)

2022-12-30 07:43:07.747133

2022-12-30 06:43:07.747359

<script.py> output:

2022-12-30 07:43:19.800587

2022-12-30 06:43:19.800819

**Daily XP200**

##### Exercise

##### Exercise

# Timezones

In order to work effectively with other timezones, you can use the pytz library. To use timezones, you need to import the timezone object from the pytz module. Then you can use the timezone constructor and pass it a name of a timezone, such as CT = timezone('US/Central'). You can get a full list of timezone names at [Wikipedia](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones). In Python 3, you can make a datetime object "aware" by passing a timezone as the tzinfo keyword argument to the .replace() method on a datetime instance.

An "aware" datetime object has an .astimezone() method that accepts a timezone object and returns a new datetime object in the desired timezone. If the tzinfo is not set for the datetime object it assumes the timezone of the computer you are working on.

A list, daily\_summaries, has been supplied for you it contains the datetime and rail ridership for trains going to New York. You need to determine the time in New York so you can align it with the New York Transit Authority data.

##### Instructions

**100 XP**

* Create a Timezone object for Chicago ('US/Central') called chicago\_usa\_tz.
* Create a Timezone object for New York ('US/Eastern') called ny\_usa\_tz.
* Iterate over the daily\_summaries, unpacking it into the variables orig\_dt and ridership.
  + Make the orig\_dt timezone "aware" for Chicago, using chicago\_usa\_tz. Store the result in chicago\_dt.
  + Convert chicago\_dt to the New York timezone, ny\_dt.
  + Print the chicago\_dt, ny\_dt, and ridership.

# Create a Timezone object for Chicago

chicago\_usa\_tz = \_\_\_\_

# Create a Timezone object for New York

ny\_usa\_tz = \_\_\_\_

# Iterate over the daily\_summaries list

for \_\_\_\_, \_\_\_\_ in \_\_\_\_:

    # Make the orig\_dt timezone "aware" for Chicago

    chicago\_dt = \_\_\_\_

    # Convert chicago\_dt to the New York Timezone

    ny\_dt = \_\_\_\_

    # Print the chicago\_dt, ny\_dt, and ridership

    print('Chicago: %s, NY: %s, Ridership: %s' % (chicago\_dt, ny\_dt, ridership))

# Compute the local datetime: local\_dt local\_dt = datetime.now() # Print the local datetime print(local\_dt) # Compute the UTC datetime: utc\_dt utc\_dt = datetime.utcnow() # Print the UTC datetime print(utc\_dt)

# Create a Timezone object for Chicago

chicago\_usa\_tz = timezone(('US/Central'))

# Create a Timezone object for New York

ny\_usa\_tz = timezone('US/Eastern')

# Iterate over the daily\_summaries list

for orig\_dt, ridership in daily\_summaries:

    # Make the orig\_dt timezone "aware" for Chicago

    chicago\_dt = orig\_dt.replace(tzinfo=chicago\_usa\_tz)

    # Convert chicago\_dt to the New York Timezone

    ny\_dt = chicago\_dt.astimezone(ny\_usa\_tz)

    # Print the chicago\_dt, ny\_dt, and ridership

    print('Chicago: %s, NY: %s, Ridership: %s' % (chicago\_dt, ny\_dt, ridership))

<script.py> output: Chicago: 2001-01-01 21:11:00-05:51, NY: 2001-01-01 22:02:00-05:00, Ridership: 126455 Chicago: 2001-01-02 13:19:00-05:51, NY: 2001-01-02 14:10:00-05:00, Ridership: 501952 Chicago: 2001-01-03 17:42:00-05:51, NY: 2001-01-03 18:33:00-05:00, Ridership: 536432 Chicago: 2001-01-04 22:17:00-05:51, NY: 2001-01-04 23:08:00-05:00, Ridership: 550011 Chicago: 2001-01-05 07:01:00-05:51, NY: 2001-01-05 07:52:00-05:00, Ridership: 557917 Chicago: 2001-01-06 03:57:00-05:51, NY: 2001-01-06 04:48:00-05:00, Ridership: 255356 Chicago: 2001-01-07 15:29:00-05:51, NY: 2001-01-07 16:20:00-05:00, Ridership: 169825 Chicago: 2001-01-08 19:55:00-05:51, NY: 2001-01-08 20:46:00-05:00, Ridership: 590706 Chicago: 2001-01-09 03:13:00-05:51, NY: 2001-01-09 04:04:00-05:00, Ridership: 599905

**Daily XP300**

**Time Travel (Adding and Subtracting Time)**

**50 XP**

**1. Time Travel (Adding and Subtracting Time)**

Another common thing to do with time object is to peer into the future or past to find data. Let's look at how that works.

**2. Incrementing through time**

A very common case when working with times is to get a date 30, 60, 90 days in the past from some date. In Python we use the timedelta object from the datetime module to represent differences in datetime objects. You can create a timedelta by passing any number of keyword arguments such as days, seconds, microseconds, milliseconds, minutes, hours, and weeks to it. Once we have a timedelta object, we can add or subtract it from a datetime object to get a datetime object relative to the original datetime object. Let's look at how this works in practice. First, we import timedelta from the datetime module. Next we'll make a timedelta instance called flashback for 90 days. Then, we can see the starting point by printing the datetime we'll be working with.

**3. Adding and subtracting timedeltas**

Then I'll print the the date minus 90 days and finally plus 90 days. This can be useful to compare a date this year to one from the prior year, compare by quarter as we did here, or compare month to month. Let's look at how we can find the time between two dates.

**4. Datetime differences**

Just like we were able to subtract a timedelta from a datetime to find a date in the past, we can also calculate the difference between two dates to get the timedelta between in return. Let's find out how much time has elapsed between the first two violations in our list. I've already parsed both dates into record\_dt and record2\_dt, respectively. Now I subtract one from the other and store the result as time\_diff. This returns a timedelta, which I used the type function to demonstrate here, that represents the time difference between the two datetimes. Finally, I can print the time\_diff and see that that time difference between violations was 4 seconds! In fact after some more digging, I was able to determine that the average difference was 15-point-3 seconds! That's a lot of parking violations! Now, it's your turn to practice this.

**Daily XP350**

##### Exercise

##### Exercise

# Finding a time in the future and from the past

Another common case when working with times is to get a date 30, 60, or 90 days in the past from some date. In Python, the timedelta object from the datetime module is used to represent differences in datetime objects. You can create a timedelta by passing any number of keyword arguments such as days, seconds, microseconds, milliseconds, minutes, hours, and weeks to timedelta().

Once you have a timedelta object, you can add or subtract it from a datetime object to get a datetime object relative to the original datetime object.

A dictionary, daily\_summaries, has been supplied for you. It contains the datetime as the key with a dict as the value that has 'day\_type' and 'total\_ridership' keys. A list of datetimes to review called review\_dates is also available.

##### Instructions

**100 XP**

* Import timedelta from the datetime module.
* Build a timedelta of 30 days called glanceback using timedelta().
* Iterate over the review\_dates, using date as your iterator variable.
  + Calculate the date 30 days back by subtracting glanceback from date.
  + Print the date, along with 'day\_type' and 'total\_ridership' from daily\_summaries for that date.
  + Print the prior\_period\_dt, along with 'day\_type' and 'total\_ridership' from daily\_summaries for that date (prior\_period\_dt).
* # Import timedelta from the datetime module
* \_\_\_\_
* # Build a timedelta of 30 days: glanceback
* glanceback = \_\_\_\_
* # Iterate over the review\_dates as date
* for \_\_\_\_ in \_\_\_\_:
* # Calculate the date 30 days back: prior\_period\_dt
* prior\_period\_dt = \_\_\_\_ - \_\_\_\_
* # Print the review\_date, day\_type and total\_ridership
* print('Date: %s, Type: %s, Total Ridership: %s' %
* (\_\_\_\_,
* \_\_\_\_[\_\_\_\_]['\_\_\_\_'],
* \_\_\_\_[\_\_\_\_]['\_\_\_\_']))
* # Print the prior\_period\_dt, day\_type and total\_ridership
* print('Date: %s, Type: %s, Total Ridership: %s' %
* (\_\_\_\_,
* \_\_\_\_,
* \_\_\_\_))

# Create a Timezone object for Chicago

chicago\_usa\_tz = timezone(('US/Central'))

# Create a Timezone object for New York

ny\_usa\_tz = timezone('US/Eastern')

# Iterate over the daily\_summaries list

for orig\_dt, ridership in daily\_summaries:

# Make the orig\_dt timezone "aware" for Chicago

chicago\_dt = orig\_dt.replace(tzinfo=chicago\_usa\_tz)

# Convert chicago\_dt to the New York Timezone

ny\_dt = chicago\_dt.astimezone(ny\_usa\_tz)

# Print the chicago\_dt, ny\_dt, and ridership

print('Chicago: %s, NY: %s, Ridership: %s' % (chicago\_dt, ny\_dt, ridership))

# Import timedelta from the datetime module

from datetime import timedelta

# Build a timedelta of 30 days: glanceback

glanceback = timedelta(days=30)

# Iterate over the review\_dates as date

for date in review\_dates:

    # Calculate the date 30 days back: prior\_period\_dt

    prior\_period\_dt = date - glanceback

    # Print the review\_date, day\_type and total\_ridership

    print('Date: %s, Type: %s, Total Ridership: %s' %

         (date,

          daily\_summaries[date]['day\_type'],

          daily\_summaries[date]['total\_ridership']))

    # Print the prior\_period\_dt, day\_type and total\_ridership

    print('Date: %s, Type: %s, Total Ridership: %s' %

         (prior\_period\_dt,

          daily\_summaries[prior\_period\_dt]['day\_type'],

          daily\_summaries[prior\_period\_dt]['total\_ridership']))

# Import timedelta from the datetime module

from datetime import timedelta

# Build a timedelta of 30 days: glanceback

glanceback = timedelta(days=30)

# Iterate over the review\_dates as date

for date in review\_dates:

# Calculate the date 30 days back: prior\_period\_dt

prior\_period\_dt = date - glanceback

# Print the review\_date, day\_type and total\_ridership

print('Date: %s, Type: %s, Total Ridership: %s' %

(date,

daily\_summaries[date]['day\_type'],

daily\_summaries[date]['total\_ridership']))

# Print the prior\_period\_dt, day\_type and total\_ridership

print('Date: %s, Type: %s, Total Ridership: %s' %

(prior\_period\_dt,

daily\_summaries[prior\_period\_dt]['day\_type'],

daily\_summaries[prior\_period\_dt]['total\_ridership']))

Date: 2013-12-22 00:00:00, Type: U, Total Ridership: 685457

Date: 2013-11-22 00:00:00, Type: W, Total Ridership: 1752614

Date: 2013-12-23 00:00:00, Type: W, Total Ridership: 1236510

Date: 2013-11-23 00:00:00, Type: A, Total Ridership: 1048943

Date: 2013-12-24 00:00:00, Type: W, Total Ridership: 815873

Date: 2013-11-24 00:00:00, Type: U, Total Ridership: 674817

Date: 2013-12-25 00:00:00, Type: U, Total Ridership: 363078

Date: 2013-11-25 00:00:00, Type: W, Total Ridership: 1641025

Date: 2013-12-26 00:00:00, Type: W, Total Ridership: 995622

Date: 2013-11-26 00:00:00, Type: W, Total Ridership: 1681213

Date: 2013-12-27 00:00:00, Type: W, Total Ridership: 1191650

Date: 2013-11-27 00:00:00, Type: W, Total Ridership: 1441786

Date: 2013-12-28 00:00:00, Type: A, Total Ridership: 911223

Date: 2013-11-28 00:00:00, Type: U, Total Ridership: 554312

Date: 2013-12-29 00:00:00, Type: U, Total Ridership: 627779

Date: 2013-11-29 00:00:00, Type: W, Total Ridership: 1074544

Date: 2013-12-30 00:00:00, Type: W, Total Ridership: 1142767

Date: 2013-11-30 00:00:00, Type: A, Total Ridership: 1013178

Date: 2013-12-31 00:00:00, Type: W, Total Ridership: 116130

Date: 2013-12-01 00:00:00, Type: U, Total Ridership: 704442

<script.py> output:

Date: 2013-12-22 00:00:00, Type: U, Total Ridership: 685457

Date: 2013-11-22 00:00:00, Type: W, Total Ridership: 1752614

Date: 2013-12-23 00:00:00, Type: W, Total Ridership: 1236510

Date: 2013-11-23 00:00:00, Type: A, Total Ridership: 1048943

Date: 2013-12-24 00:00:00, Type: W, Total Ridership: 815873

Date: 2013-11-24 00:00:00, Type: U, Total Ridership: 674817

Date: 2013-12-25 00:00:00, Type: U, Total Ridership: 363078

Date: 2013-11-25 00:00:00, Type: W, Total Ridership: 1641025

Date: 2013-12-26 00:00:00, Type: W, Total Ridership: 995622

Date: 2013-11-26 00:00:00, Type: W, Total Ridership: 1681213

Date: 2013-12-27 00:00:00, Type: W, Total Ridership: 1191650

Date: 2013-11-27 00:00:00, Type: W, Total Ridership: 1441786

Date: 2013-12-28 00:00:00, Type: A, Total Ridership: 911223

Date: 2013-11-28 00:00:00, Type: U, Total Ridership: 554312

Date: 2013-12-29 00:00:00, Type: U, Total Ridership: 627779

Date: 2013-11-29 00:00:00, Type: W, Total Ridership: 1074544

Date: 2013-12-30 00:00:00, Type: W, Total Ridership: 1142767

Date: 2013-11-30 00:00:00, Type: A, Total Ridership: 1013178

Date: 2013-12-31 00:00:00, Type: W, Total Ridership: 116130

Date: 2013-12-01 00:00:00, Type: U, Total Ridership: 704442

**Daily XP450**

##### Exercise

##### Exercise

# Finding differences in DateTimes

Just like you were able to subtract a timedelta from a datetime to find a date in the past, you can also calculate the difference between two dates to get the timedelta between in return. Here, you'll find out how much time has elapsed between two transit dates.

A list of tuples called date\_ranges is provided for you. We took the dates from our dataset at every 30th record, and we paired up the records into tuples in a stepwise fashion.

##### Instructions

**100 XP**

* Iterate over date\_ranges, unpacking it into start\_date and end\_date.
  + Print the end\_date and start\_date using the same print() function.
  + Print the difference between each end\_date and start\_date.

# Iterate over the date\_ranges

for \_\_\_\_, \_\_\_\_ in \_\_\_\_:

    # Print the End and Start Date

    print(\_\_\_\_, \_\_\_\_)

    # Print the difference between each end and start date

    \_\_\_\_

# Iterate over the date\_ranges

for start\_date, end\_date in date\_ranges:

    # Print the End and Start Date

    print(end\_date, start\_date)

    # Print the difference between each end and start date

    #from datetime import datetime, timedelta

    print(end\_date - start\_date)

# Iterate over the date\_ranges

for start\_date, end\_date in date\_ranges:

# Print the End and Start Date

print(end\_date, start\_date)

# Print the difference between each end and start date

#from datetime import datetime, timedelta

print(end\_date - start\_date)

2001-04-30 00:00:00 2001-03-31 00:00:00

30 days, 0:00:00

2001-06-29 00:00:00 2001-05-30 00:00:00

30 days, 0:00:00

2001-08-28 00:00:00 2001-07-29 00:00:00

30 days, 0:00:00

2001-10-27 00:00:00 2001-09-27 00:00:00

30 days, 0:00:00

2001-12-26 00:00:00 2001-11-26 00:00:00

30 days, 0:00:00

2002-02-24 00:00:00 2002-01-25 00:00:00

30 days, 0:00:00

2002-04-25 00:00:00 2002-03-26 00:00:00

30 days, 0:00:00

2002-06-24 00:00:00 2002-05-25 00:00:00

30 days, 0:00:00

2002-08-23 00:00:00 2002-07-24 00:00:00

30 days, 0:00:00

2002-10-22 00:00:00 2002-09-22 00:00:00

30 days, 0:00:00

2002-12-21 00:00:00 2002-11-21 00:00:00

30 days, 0:00:00

2003-02-19 00:00:00 2003-01-20 00:00:00

30 days, 0:00:00

2003-04-20 00:00:00 2003-03-21 00:00:00

30 days, 0:00:00

2003-06-19 00:00:00 2003-05-20 00:00:00

30 days, 0:00:00

2003-08-18 00:00:00 2003-07-19 00:00:00

30 days, 0:00:00

2003-10-17 00:00:00 2003-09-17 00:00:00

30 days, 0:00:00

2003-12-16 00:00:00 2003-11-16 00:00:00

30 days, 0:00:00

2004-02-14 00:00:00 2004-01-15 00:00:00

30 days, 0:00:00

2004-04-14 00:00:00 2004-03-15 00:00:00

30 days, 0:00:00

2004-06-13 00:00:00 2004-05-14 00:00:00

30 days, 0:00:00

2004-08-12 00:00:00 2004-07-13 00:00:00

30 days, 0:00:00

2004-10-11 00:00:00 2004-09-11 00:00:00

30 days, 0:00:00

2004-12-10 00:00:00 2004-11-10 00:00:00

30 days, 0:00:00

2005-02-08 00:00:00 2005-01-09 00:00:00

30 days, 0:00:00

2005-04-09 00:00:00 2005-03-10 00:00:00

30 days, 0:00:00

2005-06-08 00:00:00 2005-05-09 00:00:00

30 days, 0:00:00

2005-08-07 00:00:00 2005-07-08 00:00:00

30 days, 0:00:00

2005-10-06 00:00:00 2005-09-06 00:00:00

30 days, 0:00:00

2005-12-05 00:00:00 2005-11-05 00:00:00

30 days, 0:00:00

2006-02-03 00:00:00 2006-01-04 00:00:00

30 days, 0:00:00

2006-04-04 00:00:00 2006-03-05 00:00:00

30 days, 0:00:00

2006-06-03 00:00:00 2006-05-04 00:00:00

30 days, 0:00:00

2006-08-02 00:00:00 2006-07-03 00:00:00

30 days, 0:00:00

2006-10-01 00:00:00 2006-09-01 00:00:00

30 days, 0:00:00

2006-11-30 00:00:00 2006-10-31 00:00:00

30 days, 0:00:00

2007-01-29 00:00:00 2006-12-30 00:00:00

30 days, 0:00:00

2007-03-30 00:00:00 2007-02-28 00:00:00

30 days, 0:00:00

2007-05-29 00:00:00 2007-04-29 00:00:00

30 days, 0:00:00

2007-07-28 00:00:00 2007-06-28 00:00:00

30 days, 0:00:00

2007-09-26 00:00:00 2007-08-27 00:00:00

30 days, 0:00:00

2007-11-25 00:00:00 2007-10-26 00:00:00

30 days, 0:00:00

2008-01-24 00:00:00 2007-12-25 00:00:00

30 days, 0:00:00

2008-03-24 00:00:00 2008-02-23 00:00:00

30 days, 0:00:00

2008-05-23 00:00:00 2008-04-23 00:00:00

30 days, 0:00:00

2008-07-22 00:00:00 2008-06-22 00:00:00

30 days, 0:00:00

2008-09-20 00:00:00 2008-08-21 00:00:00

30 days, 0:00:00

2008-11-19 00:00:00 2008-10-20 00:00:00

30 days, 0:00:00

2009-01-18 00:00:00 2008-12-19 00:00:00

30 days, 0:00:00

2009-03-19 00:00:00 2009-02-17 00:00:00

30 days, 0:00:00

2009-05-18 00:00:00 2009-04-18 00:00:00

30 days, 0:00:00

2009-07-17 00:00:00 2009-06-17 00:00:00

30 days, 0:00:00

2009-09-15 00:00:00 2009-08-16 00:00:00

30 days, 0:00:00

2009-11-14 00:00:00 2009-10-15 00:00:00

30 days, 0:00:00

2010-01-13 00:00:00 2009-12-14 00:00:00

30 days, 0:00:00

2010-03-14 00:00:00 2010-02-12 00:00:00

30 days, 0:00:00

2010-05-13 00:00:00 2010-04-13 00:00:00

30 days, 0:00:00

2010-07-12 00:00:00 2010-06-12 00:00:00

30 days, 0:00:00

2010-09-10 00:00:00 2010-08-11 00:00:00

30 days, 0:00:00

2010-11-09 00:00:00 2010-10-10 00:00:00

30 days, 0:00:00

2011-01-08 00:00:00 2010-12-09 00:00:00

30 days, 0:00:00

2011-03-09 00:00:00 2011-02-07 00:00:00

30 days, 0:00:00

2011-05-08 00:00:00 2011-04-08 00:00:00

30 days, 0:00:00

2011-07-07 00:00:00 2011-06-07 00:00:00

30 days, 0:00:00

2011-09-05 00:00:00 2011-08-06 00:00:00

30 days, 0:00:00

2011-10-04 00:00:00 2011-10-05 00:00:00

-1 day, 0:00:00

2011-12-03 00:00:00 2011-11-03 00:00:00

30 days, 0:00:00

2012-02-01 00:00:00 2012-01-02 00:00:00

30 days, 0:00:00

2012-04-01 00:00:00 2012-03-02 00:00:00

30 days, 0:00:00

2012-05-31 00:00:00 2012-05-01 00:00:00

30 days, 0:00:00

2012-07-30 00:00:00 2012-06-30 00:00:00

30 days, 0:00:00

2012-09-28 00:00:00 2012-08-29 00:00:00

30 days, 0:00:00

2012-11-27 00:00:00 2012-10-28 00:00:00

30 days, 0:00:00

2013-01-26 00:00:00 2012-12-27 00:00:00

30 days, 0:00:00

2013-03-27 00:00:00 2013-02-25 00:00:00

30 days, 0:00:00

2013-05-26 00:00:00 2013-04-26 00:00:00

30 days, 0:00:00

2013-07-25 00:00:00 2013-06-25 00:00:00

30 days, 0:00:00

2013-09-23 00:00:00 2013-08-24 00:00:00

30 days, 0:00:00

2013-11-22 00:00:00 2013-10-23 00:00:00

30 days, 0:00:00

<script.py> output:

2001-03-01 00:00:00 2001-01-30 00:00:00

30 days, 0:00:00

2001-04-30 00:00:00 2001-03-31 00:00:00

30 days, 0:00:00

2001-06-29 00:00:00 2001-05-30 00:00:00

30 days, 0:00:00

2001-08-28 00:00:00 2001-07-29 00:00:00

30 days, 0:00:00

2001-10-27 00:00:00 2001-09-27 00:00:00

30 days, 0:00:00

2001-12-26 00:00:00 2001-11-26 00:00:00

30 days, 0:00:00

2002-02-24 00:00:00 2002-01-25 00:00:00

30 days, 0:00:00

2002-04-25 00:00:00 2002-03-26 00:00:00

30 days, 0:00:00

2002-06-24 00:00:00 2002-05-25 00:00:00

30 days, 0:00:00

2002-08-23 00:00:00 2002-07-24 00:00:00

30 days, 0:00:00

2002-10-22 00:00:00 2002-09-22 00:00:00

30 days, 0:00:00

2002-12-21 00:00:00 2002-11-21 00:00:00

30 days, 0:00:00

2003-02-19 00:00:00 2003-01-20 00:00:00

30 days, 0:00:00

2003-04-20 00:00:00 2003-03-21 00:00:00

30 days, 0:00:00

2003-06-19 00:00:00 2003-05-20 00:00:00

30 days, 0:00:00

2003-08-18 00:00:00 2003-07-19 00:00:00

30 days, 0:00:00

2003-10-17 00:00:00 2003-09-17 00:00:00

30 days, 0:00:00

2003-12-16 00:00:00 2003-11-16 00:00:00

30 days, 0:00:00

2004-02-14 00:00:00 2004-01-15 00:00:00

30 days, 0:00:00

2004-04-14 00:00:00 2004-03-15 00:00:00

30 days, 0:00:00

2004-06-13 00:00:00 2004-05-14 00:00:00

30 days, 0:00:00

2004-08-12 00:00:00 2004-07-13 00:00:00

30 days, 0:00:00

2004-10-11 00:00:00 2004-09-11 00:00:00

30 days, 0:00:00

2004-12-10 00:00:00 2004-11-10 00:00:00

30 days, 0:00:00

2005-02-08 00:00:00 2005-01-09 00:00:00

30 days, 0:00:00

2005-04-09 00:00:00 2005-03-10 00:00:00

30 days, 0:00:00

2005-06-08 00:00:00 2005-05-09 00:00:00

30 days, 0:00:00

2005-08-07 00:00:00 2005-07-08 00:00:00

30 days, 0:00:00

2005-10-06 00:00:00 2005-09-06 00:00:00

30 days, 0:00:00

2005-12-05 00:00:00 2005-11-05 00:00:00

30 days, 0:00:00

2006-02-03 00:00:00 2006-01-04 00:00:00

30 days, 0:00:00

2006-04-04 00:00:00 2006-03-05 00:00:00

30 days, 0:00:00

2006-06-03 00:00:00 2006-05-04 00:00:00

30 days, 0:00:00

2006-08-02 00:00:00 2006-07-03 00:00:00

30 days, 0:00:00

2006-10-01 00:00:00 2006-09-01 00:00:00

30 days, 0:00:00

2006-11-30 00:00:00 2006-10-31 00:00:00

30 days, 0:00:00

2007-01-29 00:00:00 2006-12-30 00:00:00

30 days, 0:00:00

2007-03-30 00:00:00 2007-02-28 00:00:00

30 days, 0:00:00

2007-05-29 00:00:00 2007-04-29 00:00:00

30 days, 0:00:00

2007-07-28 00:00:00 2007-06-28 00:00:00

30 days, 0:00:00

2007-09-26 00:00:00 2007-08-27 00:00:00

30 days, 0:00:00

2007-11-25 00:00:00 2007-10-26 00:00:00

30 days, 0:00:00

2008-01-24 00:00:00 2007-12-25 00:00:00

30 days, 0:00:00

2008-03-24 00:00:00 2008-02-23 00:00:00

30 days, 0:00:00

2008-05-23 00:00:00 2008-04-23 00:00:00

30 days, 0:00:00

2008-07-22 00:00:00 2008-06-22 00:00:00

30 days, 0:00:00

2008-09-20 00:00:00 2008-08-21 00:00:00

30 days, 0:00:00

2008-11-19 00:00:00 2008-10-20 00:00:00

30 days, 0:00:00

2009-01-18 00:00:00 2008-12-19 00:00:00

30 days, 0:00:00

2009-03-19 00:00:00 2009-02-17 00:00:00

30 days, 0:00:00

2009-05-18 00:00:00 2009-04-18 00:00:00

30 days, 0:00:00

2009-07-17 00:00:00 2009-06-17 00:00:00

30 days, 0:00:00

2009-09-15 00:00:00 2009-08-16 00:00:00

30 days, 0:00:00

2009-11-14 00:00:00 2009-10-15 00:00:00

30 days, 0:00:00

2010-01-13 00:00:00 2009-12-14 00:00:00

30 days, 0:00:00

2010-03-14 00:00:00 2010-02-12 00:00:00

30 days, 0:00:00

2010-05-13 00:00:00 2010-04-13 00:00:00

30 days, 0:00:00

2010-07-12 00:00:00 2010-06-12 00:00:00

30 days, 0:00:00

2010-09-10 00:00:00 2010-08-11 00:00:00

30 days, 0:00:00

2010-11-09 00:00:00 2010-10-10 00:00:00

30 days, 0:00:00

2011-01-08 00:00:00 2010-12-09 00:00:00

30 days, 0:00:00

2011-03-09 00:00:00 2011-02-07 00:00:00

30 days, 0:00:00

2011-05-08 00:00:00 2011-04-08 00:00:00

30 days, 0:00:00

2011-07-07 00:00:00 2011-06-07 00:00:00

30 days, 0:00:00

2011-09-05 00:00:00 2011-08-06 00:00:00

30 days, 0:00:00

2011-10-04 00:00:00 2011-10-05 00:00:00

-1 day, 0:00:00

2011-12-03 00:00:00 2011-11-03 00:00:00

30 days, 0:00:00

2012-02-01 00:00:00 2012-01-02 00:00:00

30 days, 0:00:00

2012-04-01 00:00:00 2012-03-02 00:00:00

30 days, 0:00:00

2012-05-31 00:00:00 2012-05-01 00:00:00

30 days, 0:00:00

2012-07-30 00:00:00 2012-06-30 00:00:00

30 days, 0:00:00

2012-09-28 00:00:00 2012-08-29 00:00:00

30 days, 0:00:00

2012-11-27 00:00:00 2012-10-28 00:00:00

30 days, 0:00:00

2013-01-26 00:00:00 2012-12-27 00:00:00

30 days, 0:00:00

2013-03-27 00:00:00 2013-02-25 00:00:00

30 days, 0:00:00

2013-05-26 00:00:00 2013-04-26 00:00:00

30 days, 0:00:00

2013-07-25 00:00:00 2013-06-25 00:00:00

30 days, 0:00:00

2013-09-23 00:00:00 2013-08-24 00:00:00

30 days, 0:00:00

2013-11-22 00:00:00 2013-10-23 00:00:00

30 days, 0:00:00

* Slides

# Import timedelta from the datetime module

from datetime import timedelta

# Build a timedelta of 30 days: glanceback

glanceback = timedelta(days=30)

# Iterate over the review\_dates as date

for date in review\_dates:

# Calculate the date 30 days back: prior\_period\_dt

prior\_period\_dt = date - glanceback

# Print the review\_date, day\_type and total\_ridership

print('Date: %s, Type: %s, Total Ridership: %s' %

(date,

daily\_summaries[date]['day\_type'],

daily\_summaries[date]['total\_ridership']))

# Print the prior\_period\_dt, day\_type and total\_ridership

print('Date: %s, Type: %s, Total Ridership: %s' %

(prior\_period\_dt,

daily\_summaries[prior\_period\_dt]['day\_type'],

daily\_summaries[prior\_period\_dt]['total\_ridership']))

## 1. HELP! Libraries to make it easier

There are several third-party libraries that make parsing, converting, and working with dates and times easier. One of most popular is the Pendulum library. Let's use Pendulum here to do some common datetime operations.

## 2. Parsing time with pendulum

Pendulum provides a powerful way to convert strings to pendulum datetime objects via the dot parse() method. Just pass it a date string and it will attempt to convert into a valid pendulum datetime. Here is a great example of how helpful parse can be. In the raw data for the parking violations we have been working with this chapter, it contains the date and time in separate columns, and the AM/PM indicator is just a single character. So I start by importing pendulum. Using the first parking violation, I'm building a string called occurred that is the date plus a space plus the time and appending an M to complete the AM/PM indicator. Then, I use pendulum dot parse() on the string and instruct pendulum that it's in the Eastern timezome all on one line! Finally, I print the date that pendulum parsed. (Note that they use a aware version os the ISO standard string). If we were to do this with datetime, we'd need to use strptime with a format string, and use the replace method to fix the timezone. Let's look at another place where pendulum is helpful - timezones.

## 3. Timezone hopping with pendulum

Pendulum has wonderful support for timezones, and comes with the Olsen Database that I mentioned earlier built into it. It provides an in\_timezone() method that can be used to convert a pendulum object to a desired timezone. Also, Pendulum's now method accepts a timezone so you can generate the current time easily for any location in the world. I saved the results for our pendulum parsing example into the violations\_dts list as you can see here.

## 4. More timezone hopping

I'm going to iterate over the records and convert them all the to Tokyo timezone. Finally, I'm going to use the now() method with the Tokyo timezone to get the current time in Tokyo.

## 5. Humanizing differences

Pendulum has an alternative to timedelta called a period when calculating the difference between two dates by subtraction that provides methods such as dot in\_days/weeks/hour/minutes and in\_words to output in a chosen manner. Let's take two of our violation datetimes and calculate the difference. Here you can see the period object where we got a timedelta previously. Next I use the in\_words() method to get a nice English representation of the difference.

## 6. More human than human

Pendulum provides the ability to set a locale and get it in other languages. Then, I use the in\_days() method to see the difference in days and finally I use in\_hours to see the difference in hours. This is just a taste of what pendulum can do. It would be well worth your time after this class to play more with pendulum and incorporate it into your work. Your turn to take it for a spin.

## 7. Let's practice!

**Daily XP500**

##### Exercise

##### Exercise

# Localizing time with pendulum

Here, you're going to use pendulum to practice doing some common datetime operations!

##### Instructions

**100 XP**

* Import the pendulum module.
* Create a now datetime for Tokyo ('Asia/Tokyo') called tokyo\_dt.
* Convert tokyo\_dt to Los Angeles time ('America/Los\_Angeles'). Store the result as la\_dt.
* Print the ISO 8601 string of la\_dt, using the .to\_iso8601\_string() method.

# Import the pendulum module

\_\_\_\_

# Create a now datetime for Tokyo: tokyo\_dt

tokyo\_dt = \_\_\_\_

# Covert the tokyo\_dt to Los Angeles: la\_dt

la\_dt = \_\_\_\_

# Print the ISO 8601 string of la\_dt

print(\_\_\_\_)

# Import the pendulum module

import pendulum

# Create a now datetime for Tokyo: tokyo\_dt

tokyo\_dt = pendulum.now('Asia/Tokyo')

# Covert the tokyo\_dt to Los Angeles: la\_dt

la\_dt = tokyo\_dt.in\_timezone('America/Los\_Angeles')

# Print the ISO 8601 string of la\_dt

print(la\_dt.to\_iso8601\_string())

# Import the pendulum module

import pendulum

# Create a now datetime for Tokyo: tokyo\_dt

tokyo\_dt = pendulum.now('Asia/Tokyo')

# Covert the tokyo\_dt to Los Angeles: la\_dt

la\_dt = tokyo\_dt.in\_timezone('America/Los\_Angeles')

# Print the ISO 8601 string of la\_dt

print(la\_dt.to\_iso8601\_string())

2022-12-30T13:35:44.523343-08:00

<script.py> output:

2022-12-30T13:36:18.374122-08:00

**Daily XP600**

##### Exercise

##### Exercise

# Humanizing Differences with Pendulum

Pendulum provides a powerful way to convert strings to pendulum datetime objects via the .parse() method. Just pass it a date string and it will attempt to convert into a valid pendulum datetime. By default, .parse() can process dates in ISO 8601 format. To allow it to parse other date formats, pass strict = False.

It also has a wonderful alternative to timedelta. When calculating the difference between two dates by subtraction, pendulum provides methods such as .in\_days() to output the difference in a chosen metric. These are just the beginning of what pendulum can do for you.

A list of tuples called date\_ranges is provided for you. This is the same list of tuples that contain two dates that was used a few exercises prior. You'll be focusing on comparing ranges of records.

You can learn more in the [pendulum documentation](https://pendulum.eustace.io/docs/). Here, it has been imported for you.

##### Instructions

**100 XP**

* Iterate over the date\_ranges list, unpacking it into start\_date and end\_date. These dates are not in ISO 8601 format.
* Use pendulum to convert the start\_date string to a pendulum date called start\_dt.
* Use pendulum to convert the end\_date string to pendulum date called end\_dt.
* Calculate the difference between end\_dt and start\_dt. Store the result as diff\_period.
* Print the difference in days, using the .in\_days() method.

# Iterate over date\_ranges

for \_\_\_\_, \_\_\_\_ in \_\_\_\_:

    # Convert the start\_date string to a pendulum date: start\_dt

    start\_dt = \_\_\_\_

    # Convert the end\_date string to a pendulum date: end\_dt

    end\_dt = \_\_\_\_

    # Print the End and Start Date

    print(end\_dt, start\_dt)

    # Calculate the difference between end\_dt and start\_dt: diff\_period

    diff\_period = \_\_\_\_ - \_\_\_\_

    # Print the difference in days

    \_\_\_\_

# Import the pendulum module

import pendulum

# Create a now datetime for Tokyo: tokyo\_dt

tokyo\_dt = pendulum.now('Asia/Tokyo')

# Covert the tokyo\_dt to Los Angeles: la\_dt

la\_dt = tokyo\_dt.in\_timezone('America/Los\_Angeles')

# Print the ISO 8601 string of la\_dt

print(la\_dt.to\_iso8601\_string())

# Iterate over date\_ranges

for start\_date, end\_date in date\_ranges:

    # Convert the start\_date string to a pendulum date: start\_dt

    start\_dt = pendulum.parse(start\_date, strict = False)

    # Convert the end\_date string to a pendulum date: end\_dt

    end\_dt = pendulum.parse(end\_date, strict = False)

    # Print the End and Start Date

    print(end\_dt, start\_dt)

    # Calculate the difference between end\_dt and start\_dt: diff\_period

    diff\_period = end\_dt - start\_dt

    # Print the difference in days

    print(diff\_period.in\_days())

# Iterate over date\_ranges

for start\_date, end\_date in date\_ranges:

# Convert the start\_date string to a pendulum date: start\_dt

start\_dt = pendulum.parse(start\_date, strict = False)

# Convert the end\_date string to a pendulum date: end\_dt

end\_dt = pendulum.parse(end\_date, strict = False)

# Print the End and Start Date

print(end\_dt, start\_dt)

# Calculate the difference between end\_dt and start\_dt: diff\_period

diff\_period = end\_dt - start\_dt

# Print the difference in days

print(diff\_period.in\_days())

<script.py> output:

2001-03-01T00:00:00+00:00 2001-01-30T00:00:00+00:00

30

2001-04-30T00:00:00+00:00 2001-03-31T00:00:00+00:00

30

2001-06-29T00:00:00+00:00 2001-05-30T00:00:00+00:00

30

2001-08-28T00:00:00+00:00 2001-07-29T00:00:00+00:00

30

2001-10-27T00:00:00+00:00 2001-09-27T00:00:00+00:00

30

2001-12-26T00:00:00+00:00 2001-11-26T00:00:00+00:00

30

2002-02-24T00:00:00+00:00 2002-01-25T00:00:00+00:00

30

2002-04-25T00:00:00+00:00 2002-03-26T00:00:00+00:00

30

2002-06-24T00:00:00+00:00 2002-05-25T00:00:00+00:00

30

2002-08-23T00:00:00+00:00 2002-07-24T00:00:00+00:00

30

2002-10-22T00:00:00+00:00 2002-09-22T00:00:00+00:00

30

2002-12-21T00:00:00+00:00 2002-11-21T00:00:00+00:00

30

2003-02-19T00:00:00+00:00 2003-01-20T00:00:00+00:00

30

2003-04-20T00:00:00+00:00 2003-03-21T00:00:00+00:00

30

2003-06-19T00:00:00+00:00 2003-05-20T00:00:00+00:00

30

2003-08-18T00:00:00+00:00 2003-07-19T00:00:00+00:00

30

2003-10-17T00:00:00+00:00 2003-09-17T00:00:00+00:00

30

2003-12-16T00:00:00+00:00 2003-11-16T00:00:00+00:00

30

2004-02-14T00:00:00+00:00 2004-01-15T00:00:00+00:00

30

2004-04-14T00:00:00+00:00 2004-03-15T00:00:00+00:00

30

2004-06-13T00:00:00+00:00 2004-05-14T00:00:00+00:00

30

2004-08-12T00:00:00+00:00 2004-07-13T00:00:00+00:00

30

2004-10-11T00:00:00+00:00 2004-09-11T00:00:00+00:00

30

2004-12-10T00:00:00+00:00 2004-11-10T00:00:00+00:00

30

2005-02-08T00:00:00+00:00 2005-01-09T00:00:00+00:00

30

2005-04-09T00:00:00+00:00 2005-03-10T00:00:00+00:00

30

2005-06-08T00:00:00+00:00 2005-05-09T00:00:00+00:00

30

2005-08-07T00:00:00+00:00 2005-07-08T00:00:00+00:00

30

2005-10-06T00:00:00+00:00 2005-09-06T00:00:00+00:00

30

2005-12-05T00:00:00+00:00 2005-11-05T00:00:00+00:00

30

2006-02-03T00:00:00+00:00 2006-01-04T00:00:00+00:00

30

2006-04-04T00:00:00+00:00 2006-03-05T00:00:00+00:00

30

2006-06-03T00:00:00+00:00 2006-05-04T00:00:00+00:00

30

2006-08-02T00:00:00+00:00 2006-07-03T00:00:00+00:00

30

2006-10-01T00:00:00+00:00 2006-09-01T00:00:00+00:00

30

2006-11-30T00:00:00+00:00 2006-10-31T00:00:00+00:00

30

2007-01-29T00:00:00+00:00 2006-12-30T00:00:00+00:00

30

2007-03-30T00:00:00+00:00 2007-02-28T00:00:00+00:00

30

2007-05-29T00:00:00+00:00 2007-04-29T00:00:00+00:00

30

2007-07-28T00:00:00+00:00 2007-06-28T00:00:00+00:00

30

2007-09-26T00:00:00+00:00 2007-08-27T00:00:00+00:00

30

2007-11-25T00:00:00+00:00 2007-10-26T00:00:00+00:00

30

2008-01-24T00:00:00+00:00 2007-12-25T00:00:00+00:00

30

2008-03-24T00:00:00+00:00 2008-02-23T00:00:00+00:00

30

2008-05-23T00:00:00+00:00 2008-04-23T00:00:00+00:00

30

2008-07-22T00:00:00+00:00 2008-06-22T00:00:00+00:00

30

2008-09-20T00:00:00+00:00 2008-08-21T00:00:00+00:00

30

2008-11-19T00:00:00+00:00 2008-10-20T00:00:00+00:00

30

2009-01-18T00:00:00+00:00 2008-12-19T00:00:00+00:00

30

2009-03-19T00:00:00+00:00 2009-02-17T00:00:00+00:00

30

2009-05-18T00:00:00+00:00 2009-04-18T00:00:00+00:00

30

2009-07-17T00:00:00+00:00 2009-06-17T00:00:00+00:00

30

2009-09-15T00:00:00+00:00 2009-08-16T00:00:00+00:00

30

2009-11-14T00:00:00+00:00 2009-10-15T00:00:00+00:00

30

2010-01-13T00:00:00+00:00 2009-12-14T00:00:00+00:00

30

2010-03-14T00:00:00+00:00 2010-02-12T00:00:00+00:00

30

2010-05-13T00:00:00+00:00 2010-04-13T00:00:00+00:00

30

2010-07-12T00:00:00+00:00 2010-06-12T00:00:00+00:00

30

2010-09-10T00:00:00+00:00 2010-08-11T00:00:00+00:00

30

2010-11-09T00:00:00+00:00 2010-10-10T00:00:00+00:00

30

2011-01-08T00:00:00+00:00 2010-12-09T00:00:00+00:00

30

2011-03-09T00:00:00+00:00 2011-02-07T00:00:00+00:00

30

2011-05-08T00:00:00+00:00 2011-04-08T00:00:00+00:00

30

2011-07-07T00:00:00+00:00 2011-06-07T00:00:00+00:00

30

2011-09-05T00:00:00+00:00 2011-08-06T00:00:00+00:00

30

2011-10-04T00:00:00+00:00 2011-10-05T00:00:00+00:00

-1

2011-12-03T00:00:00+00:00 2011-11-03T00:00:00+00:00

30

2012-02-01T00:00:00+00:00 2012-01-02T00:00:00+00:00

30

2012-04-01T00:00:00+00:00 2012-03-02T00:00:00+00:00

30

2012-05-31T00:00:00+00:00 2012-05-01T00:00:00+00:00

30

2012-07-30T00:00:00+00:00 2012-06-30T00:00:00+00:00

30

2012-09-28T00:00:00+00:00 2012-08-29T00:00:00+00:00

30

2012-11-27T00:00:00+00:00 2012-10-28T00:00:00+00:00

30

2013-01-26T00:00:00+00:00 2012-12-27T00:00:00+00:00

30

2013-03-27T00:00:00+00:00 2013-02-25T00:00:00+00:00

30

2013-05-26T00:00:00+00:00 2013-04-26T00:00:00+00:00

30

2013-07-25T00:00:00+00:00 2013-06-25T00:00:00+00:00

30

2013-09-23T00:00:00+00:00 2013-08-24T00:00:00+00:00

30

2013-11-22T00:00:00+00:00 2013-10-23T00:00:00+00:00

30

**Daily XP800**

**Counting within Date Ranges**

**50 XP**

**1. Case Study - Counting Crimes**

Now it's time to put your learning to practical use with a case study on the Chicago Crime Data.

**2. Data Set Overview**

Our data is in a CSV file and looks like you see here. It contains the details for each crime in the City of Chicago. It's worth noting that I've shrunk the dataset down to be more manageable via sampling so these don't represent every crime that occurred. The full dataset is available on the City of Chicago's Open Data Portal.

**3. Part 1 - Step 1**

You'll begin this case study by reading data from a CSV and building a list to hold your data.

**4. Part 1 - Step 2**

In the second step, you'll count the data by month using a counter similarly to how we did previously, but with a small twist, which I'll come back to in a second. Then you'll use the month date part to group/count the data by month. So about that twist. In the date part grouping example we used a defaultdict. Here you'll be using a counter. Hint: both work the same way since both are based on dictionaries.

**5. Part 1 - Step 3**

Next, we want to extract the data into a dictionary keys by month that stores a list of the location types where the crimes occurred that month. We'll use the defaultdict we learned about in when working on eateries coupled with the date component grouping we just used in the prior step and learned earlier.

**6. Part 1 - Final**

So to answer the real question, "What are the five most common crime locations per month?" We'll use a Counter on our new dictionary as we did previous to find the answer we're seeking. Good luck with part 1!

**Daily XP850**

##### Exercise

##### Exercise

# Reading your data with CSV Reader and Establishing your Data Containers

Let's get started! The exercises in this chapter are intentionally more challenging, to give you a chance to really solidify your knowledge. Don't lose heart if you find yourself stuck; think back to the concepts you've learned in previous chapters and how you can apply them to this crime dataset. Good luck!

Your data file, crime\_sampler.csv contains the date (1st column), block where it occurred (2nd column), primary type of the crime (3rd), description of the crime (4th), description of the location (5th), if an arrest was made (6th), was it a domestic case (7th), and city district (8th).

Here, however, you'll focus only 4 columns: The date, type of crime, location, and whether or not the crime resulted in an arrest.

Your job in this exercise is to use a CSV Reader to load up a list to hold the data you're going to analyze.

##### Instructions

**100 XP**

* Import the Python csv module.
* Create a Python file object in read mode for crime\_sampler.csv called csvfile.
* Create an empty list called crime\_data.
* Loop over a csv reader on the file object :
  + Inside the loop, append the date (first element), type of crime (third element), location description (fifth element), and arrest (sixth element) to the crime\_data list.
* Remove the first element (headers) from the crime\_data list.
* Print the first 10 records of the crime\_data list. This has been done for you, so hit 'Submit Answer' to see the result!

# Import the csv module

\_\_\_\_

# Create the file object: csvfile

csvfile = \_\_\_\_

# Create an empty list: crime\_data

crime\_data = \_\_\_\_

# Loop over a csv reader on the file object

for row in \_\_\_\_:

    # Append the date, type of crime, location description, and arrest

    crime\_data.append((row[0], row[2], row[4], row[5]))

# Remove the first element from crime\_data

\_\_\_\_

# Print the first 10 records

print(crime\_data[:10])

* IPython Shell
* Slides

# Iterate over date\_ranges

for start\_date, end\_date in date\_ranges:

# Convert the start\_date string to a pendulum date: start\_dt

start\_dt = pendulum.parse(start\_date, strict = False)

# Convert the end\_date string to a pendulum date: end\_dt

end\_dt = pendulum.parse(end\_date, strict = False)

# Print the End and Start Date

print(end\_dt, start\_dt)

# Calculate the difference between end\_dt and start\_dt: diff\_period

diff\_period = end\_dt - start\_dt

# Print the difference in days

print(diff\_period.in\_days())

# Import the csv module

import csv

# Create the file object: csvfile

csvfile = open('crime\_sampler.csv', 'r')

# Create an empty list: crime\_data

crime\_data = []

# Loop over a csv reader on the file object

for row in csv.reader(csvfile):

    # Append the date, type of crime, location description, and arrest

    crime\_data.append((row[0], row[2], row[4], row[5]))

# Remove the first element from crime\_data

crime\_data.pop(0)

# Print the first 10 records

print(crime\_data[:10])

# Import the csv module

import csv

# Create the file object: csvfile

csvfile = open('crime\_sampler.csv', 'r')

# Create an empty list: crime\_data

crime\_data = []

# Loop over a csv reader on the file object

for row in csv.reader(csvfile):

# Append the date, type of crime, location description, and arrest

crime\_data.append((row[0], row[2], row[4], row[5]))

# Remove the first element from crime\_data

crime\_data.pop()

# Print the first 10 records

print(crime\_data[:10])

[('Date', 'Primary Type', 'Location Description', 'Arrest'), ('05/23/2016 05:35:00 PM', 'ASSAULT', 'STREET', 'false'), ('03/26/2016 08:20:00 PM', 'BURGLARY', 'SMALL RETAIL STORE', 'false'), ('04/25/2016 03:05:00 PM', 'THEFT', 'DEPARTMENT STORE', 'true'), ('04/26/2016 05:30:00 PM', 'BATTERY', 'SIDEWALK', 'false'), ('06/19/2016 01:15:00 AM', 'BATTERY', 'SIDEWALK', 'false'), ('05/28/2016 08:00:00 PM', 'BATTERY', 'GAS STATION', 'false'), ('07/03/2016 03:43:00 PM', 'THEFT', 'OTHER', 'false'), ('06/11/2016 06:55:00 PM', 'PUBLIC PEACE VIOLATION', 'STREET', 'true'), ('10/04/2016 10:20:00 AM', 'BATTERY', 'STREET', 'true')]

<script.py> output:

[('Date', 'Primary Type', 'Location Description', 'Arrest'), ('05/23/2016 05:35:00 PM', 'ASSAULT', 'STREET', 'false'), ('03/26/2016 08:20:00 PM', 'BURGLARY', 'SMALL RETAIL STORE', 'false'), ('04/25/2016 03:05:00 PM', 'THEFT', 'DEPARTMENT STORE', 'true'), ('04/26/2016 05:30:00 PM', 'BATTERY', 'SIDEWALK', 'false'), ('06/19/2016 01:15:00 AM', 'BATTERY', 'SIDEWALK', 'false'), ('05/28/2016 08:00:00 PM', 'BATTERY', 'GAS STATION', 'false'), ('07/03/2016 03:43:00 PM', 'THEFT', 'OTHER', 'false'), ('06/11/2016 06:55:00 PM', 'PUBLIC PEACE VIOLATION', 'STREET', 'true'), ('10/04/2016 10:20:00 AM', 'BATTERY', 'STREET', 'true')]

# Import the csv module

import csv

# Create the file object: csvfile

csvfile = open('crime\_sampler.csv', 'r')

# Create an empty list: crime\_data

crime\_data = []

# Loop over a csv reader on the file object

for row in csv.reader(csvfile):

# Append the date, type of crime, location description, and arrest

crime\_data.append((row[0], row[2], row[4], row[5]))

# Remove the first element from crime\_data

crime\_data.pop(0)

# Print the first 10 records

print(crime\_data[:10])

[('05/23/2016 05:35:00 PM', 'ASSAULT', 'STREET', 'false'), ('03/26/2016 08:20:00 PM', 'BURGLARY', 'SMALL RETAIL STORE', 'false'), ('04/25/2016 03:05:00 PM', 'THEFT', 'DEPARTMENT STORE', 'true'), ('04/26/2016 05:30:00 PM', 'BATTERY', 'SIDEWALK', 'false'), ('06/19/2016 01:15:00 AM', 'BATTERY', 'SIDEWALK', 'false'), ('05/28/2016 08:00:00 PM', 'BATTERY', 'GAS STATION', 'false'), ('07/03/2016 03:43:00 PM', 'THEFT', 'OTHER', 'false'), ('06/11/2016 06:55:00 PM', 'PUBLIC PEACE VIOLATION', 'STREET', 'true'), ('10/04/2016 10:20:00 AM', 'BATTERY', 'STREET', 'true'), ('02/14/2017 09:00:00 PM', 'CRIMINAL DAMAGE', 'PARK PROPERTY', 'false')]

<script.py> output:

[('05/23/2016 05:35:00 PM', 'ASSAULT', 'STREET', 'false'), ('03/26/2016 08:20:00 PM', 'BURGLARY', 'SMALL RETAIL STORE', 'false'), ('04/25/2016 03:05:00 PM', 'THEFT', 'DEPARTMENT STORE', 'true'), ('04/26/2016 05:30:00 PM', 'BATTERY', 'SIDEWALK', 'false'), ('06/19/2016 01:15:00 AM', 'BATTERY', 'SIDEWALK', 'false'), ('05/28/2016 08:00:00 PM', 'BATTERY', 'GAS STATION', 'false'), ('07/03/2016 03:43:00 PM', 'THEFT', 'OTHER', 'false'), ('06/11/2016 06:55:00 PM', 'PUBLIC PEACE VIOLATION', 'STREET', 'true'), ('10/04/2016 10:20:00 AM', 'BATTERY', 'STREET', 'true'), ('02/14/2017 09:00:00 PM', 'CRIMINAL DAMAGE', 'PARK PROPERTY', 'false')]

In [1]:

**Daily XP950**

##### Exercise

##### Exercise

# Find the Months with the Highest Number of Crimes

Using the crime\_data list from the prior exercise, you'll answer a common question that arises when dealing with crime data: How many crimes are committed each month?

Feel free to use the IPython Shell to explore the crime\_data list - it has been pre-loaded for you. For example, crime\_data[0][0] will show you the first column of the first row which, in this case, is the date and time time that the crime occurred.

##### Instructions

**100 XP**

* Import Counter from collections and datetime from datetime.
* Create a Counter object called crimes\_by\_month.
* Loop over the crime\_data list:
  + Using the datetime.strptime() function, convert the first element of each item into a Python Datetime Object called date.
  + Increment the counter for the month associated with this row by one. You can access the month of date using date.month.
* Print the 3 most common months for crime.

# Import necessary modules

\_\_\_\_

\_\_\_\_

# Create a Counter Object: crimes\_by\_month

crimes\_by\_month = \_\_\_\_

# Loop over the crime\_data list

for \_\_\_\_ in \_\_\_\_:

    # Convert the first element of each item into a Python Datetime Object: date

    date = \_\_\_\_(\_\_\_\_, '%m/%d/%Y %I:%M:%S %p')

    # Increment the counter for the month of the row by one

    \_\_\_\_ += 1

# Print the 3 most common months for crime

print(\_\_\_\_)

* Slides

# Import the csv module

import csv

# Create the file object: csvfile

csvfile = open('crime\_sampler.csv', 'r')

# Create an empty list: crime\_data

crime\_data = []

# Loop over a csv reader on the file object

for row in csv.reader(csvfile):

# Append the date, type of crime, location description, and arrest

crime\_data.append((row[0], row[2], row[4], row[5]))

# Remove the first element from crime\_data

crime\_data.pop(0)

# Print the first 10 records

print(crime\_data[:10])

# Import necessary modules

from collections import Counter

from datetime import datetime

# Create a Counter Object: crimes\_by\_month

crimes\_by\_month = Counter()

# Loop over the crime\_data list

for date in crime\_data:

    # Convert the first element of each item into a Python Datetime Object: date

    date = datetime.strptime(date[0], '%m/%d/%Y %I:%M:%S %p')

    # Increment the counter for the month of the row by one

    crimes\_by\_month[date.month] += 1

# Print the 3 most common months for crime

print(crimes\_by\_month.most\_common(3))

# Import necessary modules

from collections import Counter

from datetime import datetime

# Create a Counter Object: crimes\_by\_month

crimes\_by\_month = Counter()

# Loop over the crime\_data list

for date in crime\_data:

# Convert the first element of each item into a Python Datetime Object: date

date = datetime.strptime(date[0], '%m/%d/%Y %I:%M:%S %p')

# Increment the counter for the month of the row by one

crimes\_by\_month[date.month] += 1

# Print the 3 most common months for crime

print(crimes\_by\_month.most\_common(3))

[(1, 1948), (2, 1862), (7, 1257)]

<script.py> output:

[(1, 1948), (2, 1862), (7, 1257)]

**Daily XP100**

##### Exercise

##### Exercise

# Transforming your Data Containers to Month and Location

Now let's flip your crime\_data list into a dictionary keyed by month with a list of location values for each month, and filter down to the records for the year 2016. Remember you can use the shell to look at the crime\_data list, such as crime\_data[1][4] to see the location of the crime in the second item of the list (since lists start at 0).

##### Instructions

**100 XP**

* Import defaultdict from collections and datetime from datetime.
* Create a dictionary that defaults to a list called locations\_by\_month.
* Loop over the crime\_data list:
  + Convert the first element to a date object exactly like you did in the previous exercise.
  + If the year is 2016, set the key of locations\_by\_month to be the month of date and .append() the location (fifth element of row) to the values list.
* Print the dictionary. This has been done for you, so hit 'Submit Answer' to see the result!

# Import necessary modules

\_\_\_\_

\_\_\_\_

# Create a dictionary that defaults to a list: locations\_by\_month

locations\_by\_month = \_\_\_\_

# Loop over the crime\_data list

for row in \_\_\_\_:

    # Convert the first element to a date object

    date = \_\_\_\_(\_\_\_\_, '%m/%d/%Y %I:%M:%S %p')

    # If the year is 2016

    if \_\_\_\_.\_\_\_\_ == \_\_\_\_:

        # Set the dictionary key to the month and append the location (fifth element) to the values list

        \_\_\_\_

# Print the dictionary

print(locations\_by\_month)

* Shell
* Slides

# Import necessary modules

from collections import Counter

from datetime import datetime

# Create a Counter Object: crimes\_by\_month

crimes\_by\_month = Counter()

# Loop over the crime\_data list

for date in crime\_data:

# Convert the first element of each item into a Python Datetime Object: date

date = datetime.strptime(date[0], '%m/%d/%Y %I:%M:%S %p')

# Increment the counter for the month of the row by one

crimes\_by\_month[date.month] += 1

# Print the 3 most common months for crime

print(crimes\_by\_month.most\_common(3))

# Import necessary modules

from collections import defaultdict

from datetime import datetime

# Create a dictionary that defaults to a list: locations\_by\_month

locations\_by\_month = defaultdict(list)

# Loop over the crime\_data list

for row in crime\_data:

    # Convert the first element to a date object

    date = datetime.strptime(row[0], '%m/%d/%Y %I:%M:%S %p')

    # If the year is 2016

    if date.year == 2016:

        # Set the dictionary key to the month and append the location (fifth element) to the values list

        locations\_by\_month[date.month].append(row[4])

# Print the dictionary

print(locations\_by\_month)

# Import necessary modules from collections import defaultdict from datetime import datetime # Create a dictionary that defaults to a list: locations\_by\_month locations\_by\_month = defaultdict(list) # Loop over the crime\_data list for row in crime\_data: # Convert the first element to a date object date = datetime.strptime(row[0], '%m/%d/%Y %I:%M:%S %p') # If the year is 2016 if date.year == 2016: # Set the dictionary key to the month and append the location (fifth element) to the values list locations\_by\_month[date.month].append(row[4]) # Print the dictionary print(locations\_by\_month)

**Daily XP200**

##### Exercise

##### Exercise

# Find the Most Common Crimes by Location Type by Month in 2016

Using the locations\_by\_month dictionary from the prior exercise, you'll now determine common crimes by month and location type. Because your dataset is so large, it's a good idea to use Counter to look at an aspect of it in an easier to manageable size and learn more about it.

##### Instructions

**100 XP**

* Import Counter from collections.
* Loop over the items from your dictionary, using tuple expansion to unpack locations\_by\_month.items() into month and locations.
  + Make a Counter of the locations called location\_count.
  + Print the month.
  + Print the five most common crime locations.

# Import Counter from collections

\_\_\_\_

# Loop over the items from locations\_by\_month using tuple expansion of the month and locations

for \_\_\_\_, \_\_\_\_ in \_\_\_\_:

    # Make a Counter of the locations

    location\_count = \_\_\_\_

    # Print the month

    print(\_\_\_\_)

    # Print the most common location

    print(\_\_\_\_)

# Import necessary modules

from collections import defaultdict

from datetime import datetime

# Create a dictionary that defaults to a list: locations\_by\_month

locations\_by\_month = defaultdict(list)

# Loop over the crime\_data list

for row in crime\_data:

# Convert the first element to a date object

date = datetime.strptime(row[0], '%m/%d/%Y %I:%M:%S %p')

# If the year is 2016

if date.year == 2016:

# Set the dictionary key to the month and append the location (fifth element) to the values list

locations\_by\_month[date.month].append(row[4])

# Print the dictionary

print(locations\_by\_month)

Import Counter from collections

from collections import Counter

# Loop over the items from locations\_by\_month using tuple expansion of the month and locations

for month, location in locations\_by\_month.items():

    # Make a Counter of the locations

    location\_count = Counter(location)

    # Print the month

    print(month)

    # Print the most common location

    print(location\_count.most\_common(5))

# Import Counter from collections

from collections import Counter

# Loop over the items from locations\_by\_month using tuple expansion of the month and locations

for month, location in locations\_by\_month.items():

# Make a Counter of the locations

location\_count = Counter(location)

# Print the month

print(month)

# Print the most common location

print(location\_count.most\_common(5))

5

[('STREET', 241), ('RESIDENCE', 175), ('APARTMENT', 128), ('SIDEWALK', 111), ('OTHER', 41)]

3

[('STREET', 275), ('RESIDENCE', 218), ('APARTMENT', 155), ('SIDEWALK', 109), ('OTHER', 55)]

4

[('STREET', 213), ('RESIDENCE', 171), ('APARTMENT', 152), ('SIDEWALK', 96), ('OTHER', 40)]

6

[('STREET', 245), ('RESIDENCE', 164), ('APARTMENT', 159), ('SIDEWALK', 123), ('PARKING LOT/GARAGE(NON.RESID.)', 44)]

7

[('STREET', 309), ('RESIDENCE', 177), ('APARTMENT', 166), ('SIDEWALK', 125), ('OTHER', 47)]

10

[('STREET', 248), ('RESIDENCE', 206), ('APARTMENT', 122), ('SIDEWALK', 92), ('OTHER', 62)]

2

[('STREET', 419), ('RESIDENCE', 318), ('APARTMENT', 284), ('SIDEWALK', 137), ('OTHER', 86)]

12

[('STREET', 207), ('RESIDENCE', 158), ('APARTMENT', 136), ('OTHER', 47), ('SIDEWALK', 46)]

1

[('STREET', 441), ('RESIDENCE', 342), ('APARTMENT', 276), ('SIDEWALK', 138), ('OTHER', 82)]

9

[('STREET', 279), ('RESIDENCE', 183), ('APARTMENT', 144), ('SIDEWALK', 121), ('OTHER', 39)]

11

[('STREET', 236), ('RESIDENCE', 182), ('APARTMENT', 154), ('SIDEWALK', 75), ('OTHER', 41)]

8

[('STREET', 280), ('RESIDENCE', 199), ('APARTMENT', 144), ('SIDEWALK', 109), ('OTHER', 47)]

<script.py> output:

5

[('STREET', 241), ('RESIDENCE', 175), ('APARTMENT', 128), ('SIDEWALK', 111), ('OTHER', 41)]

3

[('STREET', 275), ('RESIDENCE', 218), ('APARTMENT', 155), ('SIDEWALK', 109), ('OTHER', 55)]

4

[('STREET', 213), ('RESIDENCE', 171), ('APARTMENT', 152), ('SIDEWALK', 96), ('OTHER', 40)]

6

[('STREET', 245), ('RESIDENCE', 164), ('APARTMENT', 159), ('SIDEWALK', 123), ('PARKING LOT/GARAGE(NON.RESID.)', 44)]

7

[('STREET', 309), ('RESIDENCE', 177), ('APARTMENT', 166), ('SIDEWALK', 125), ('OTHER', 47)]

10

[('STREET', 248), ('RESIDENCE', 206), ('APARTMENT', 122), ('SIDEWALK', 92), ('OTHER', 62)]

2

[('STREET', 419), ('RESIDENCE', 318), ('APARTMENT', 284), ('SIDEWALK', 137), ('OTHER', 86)]

12

[('STREET', 207), ('RESIDENCE', 158), ('APARTMENT', 136), ('OTHER', 47), ('SIDEWALK', 46)]

1

[('STREET', 441), ('RESIDENCE', 342), ('APARTMENT', 276), ('SIDEWALK', 138), ('OTHER', 82)]

9

[('STREET', 279), ('RESIDENCE', 183), ('APARTMENT', 144), ('SIDEWALK', 121), ('OTHER', 39)]

11

[('STREET', 236), ('RESIDENCE', 182), ('APARTMENT', 154), ('SIDEWALK', 75), ('OTHER', 41)]

8

[('STREET', 280), ('RESIDENCE', 199), ('APARTMENT', 144), ('SIDEWALK', 109), ('OTHER', 47)][



## 1. Case Study - Crimes by District and Differences by Block

Now let's try our hand at the second part of the case study. First, We're going to determine how many crimes occurred by district, and then look at how types of crimes differ between city blocks.

## 2. Part 2 - Step 1

Previously, we read the data directly from the csv file into a dictionary as shown here. This example shows how to use the dictreader which gives you a dictionary per row of the file, you'll need to determine how to properly build the dictionary based on the assignment instructions. Then we can use that dictionary to do things like pop out a key, as we did in a prior exercise, and store the remainder of the data under that key in another dictionary. Here is a reminder of how to pop data from a dictionary, which leaves the original dictionary with everything but that key and value still in place.

## 3. Part 2 - Step 2

Then we want to determine how many crimes occurred by district. You'll need to loop over the dictionary Pythonically and use Counter and defaultdict as we have several times in this case study. Here is an example of how we Pythonically looped over dictionaries in our videos.

## 4. Wrapping Up

In the last step of our case study, we've identified a few blocks of data we want to concentrate on and see the differences in crimes that occur in these locations. First you'll want to take a list and get a unique set of crimes for that block as we did in chapter one. Then you'll look for difference in the unique crime sets using the set difference method as we did at the end of chapter 1. This will complete the case study! Good luck!

## 5. Let's practice!

**Daily XP350**

##### Exercise

##### Exercise

# Reading your Data with DictReader and Establishing your Data Containers

Your data file, crime\_sampler.csv contains in positional order: the date, block where it occurred, primary type of the crime, description of the crime, description of the location, if an arrest was made, was it a domestic case, and city district.

You'll now use a DictReader to load up a dictionary to hold your data with the district as the key and the rest of the data in a list. The csv, defaultdict, and datetime modules have already been imported for you.

##### Instructions

**100 XP**

* Create a Python file object in read mode for crime\_sampler.csv called csvfile.
* Create a dictionary that defaults to a list called crimes\_by\_district.
* Loop over a DictReader of the CSV file:
  + Pop 'District' from each row and store it as district.
  + Append the rest of the data (row) to the district key of crimes\_by\_district.

# Create the CSV file: csvfile

csvfile = \_\_\_\_

# Create a dictionary that defaults to a list: crimes\_by\_district

crimes\_by\_district = \_\_\_\_

# Loop over a DictReader of the CSV file

for row in \_\_\_\_:

    # Pop the district from each row: district

    district = \_\_\_\_

    # Append the rest of the data to the list for proper district in crimes\_by\_district

    \_\_\_\_

# Import Counter from collections

from collections import Counter

# Loop over the items from locations\_by\_month using tuple expansion of the month and locations

for month, location in locations\_by\_month.items():

# Make a Counter of the locations

location\_count = Counter(location)

# Print the month

print(month)

# Print the most common location

print(location\_count.most\_common(5))

# Create the CSV file: csvfile

csvfile = open('crime\_sampler.csv', 'r')

# Create a dictionary that defaults to a list: crimes\_by\_district

crimes\_by\_district = defaultdict(list)

# Loop over a DictReader of the CSV file

for row in csv.DictReader(csvfile):

    # Pop the district from each row: district

    #district = crimes\_by\_district.pop(row['District'])

    district = row.pop('District')

    # Append the rest of the data to the list for proper district in crimes\_by\_district

    crimes\_by\_district[district].append(row)

# Create the CSV file: csvfile

csvfile = open('crime\_sampler.csv', 'r')

# Create a dictionary that defaults to a list: crimes\_by\_district

crimes\_by\_district = defaultdict(list)

# Loop over a DictReader of the CSV file

for row in csv.DictReader(csvfile):

# Pop the district from each row: district

#district = crimes\_by\_district.pop(row['District'])

district = row.pop('District')

# Append the rest of the data to the list for proper district in crimes\_by\_district

crimes\_by\_district[district].append(row)



**Daily XP450**

##### Exercise

##### Exercise

# Determine the Arrests by District by Year

Using your crimes\_by\_district dictionary from the previous exercise, you'll now determine the number arrests in each City district for each year. Counter is already imported for you. You'll want to use the IPython Shell to explore the crimes\_by\_district dictionary to determine how to check if an arrest was made.

##### Instructions

**100 XP**

* Loop over the crimes\_by\_district dictionary, unpacking it into the variables district and crimes.
  + Create an empty Counter object called year\_count.
  + Loop over the crimes:
    - If there was an arrest,
      * Convert crime['Date'] to a datetime object called year.
      * Add the crime to the Counter for the year, by using year as the key of year\_count.
  + Print the Counter. This has been done for you, so hit 'Submit Answer' to see the result!
* # Loop over the crimes\_by\_district using expansion as district and crimes
* for \_\_\_\_, \_\_\_\_ in \_\_\_\_:
* # Print the district
* print(district)
* # Create an empty Counter object: year\_count
* year\_count = \_\_\_\_
* # Loop over the crimes:
* for crime in \_\_\_\_:
* # If there was an arrest
* if \_\_\_\_['\_\_\_\_'] == '\_\_\_\_':
* # Convert the Date to a datetime and get the year
* year = \_\_\_\_(\_\_\_\_, '%m/%d/%Y %I:%M:%S %p').year
* # Increment the Counter for the year
* \_\_\_\_ += 1
* # Print the counter
* print(year\_count)
* IPython Shell
* Slides

# Create the CSV file: csvfile

csvfile = open('crime\_sampler.csv', 'r')

# Create a dictionary that defaults to a list: crimes\_by\_district

crimes\_by\_district = defaultdict(list)

# Loop over a DictReader of the CSV file

for row in csv.DictReader(csvfile):

# Pop the district from each row: district

#district = crimes\_by\_district.pop(row['District'])

district = row.pop('District')

# Append the rest of the data to the list for proper district in crimes\_by\_district

crimes\_by\_district[district].append(row)

**Daily XP0**

##### Exercise

##### Exercise

# Determine the Arrests by District by Year

Using your crimes\_by\_district dictionary from the previous exercise, you'll now determine the number arrests in each City district for each year. Counter is already imported for you. You'll want to use the IPython Shell to explore the crimes\_by\_district dictionary to determine how to check if an arrest was made.

##### Instructions

**100 XP**

* Loop over the crimes\_by\_district dictionary, unpacking it into the variables district and crimes.
  + Create an empty Counter object called year\_count.
  + Loop over the crimes:
    - If there was an arrest,
      * Convert crime['Date'] to a datetime object called year.
      * Add the crime to the Counter for the year, by using year as the key of year\_count.
  + Print the Counter. This has been done for you, so hit 'Submit Answer' to see the result

# Loop over the crimes\_by\_district using expansion as district and crimes

for district, crimes in crimes\_by\_district:

    # Print the district

    print(district)

    # Create an empty Counter object: year\_count

    year\_count = Counter()

    # Loop over the crimes:

    for crime in crimes:

        # If there was an arrest

        if crime['Date'] == 'Arrest':

            # Convert the Date to a datetime and get the year

            year = datetime.strptime(crime['Date'], '%m/%d/%Y %I:%M:%S %p').year

            # Increment the Counter for the year

            year\_count[year] += 1

    # Print the counter

    print(year\_count)

# Loop over the crimes\_by\_district using expansion as district and crimes

for district, crimes in crimes\_by\_district.items():

    # Print the district

    print(district)

    # Create an empty Counter object: year\_count

    year\_count = Counter()

    # Loop over the crimes:

    for crime in crimes:

        # If there was an arrest

        if crime['Arrest'] == 'true':

            # Convert the Date to a datetime and get the year

            year = datetime.strptime(crime['Date'], '%m/%d/%Y %I:%M:%S %p').year

            # Increment the Counter for the year

            year\_count[year] += 1

    # Print the counter

    print(year\_count)

14 Counter({2016: 59, 2017: 8}) 24 Counter({2016: 51, 2017: 10}) 6 Counter({2016: 157, 2017: 32}) 15 Counter({2016: 154, 2017: 16}) 12 Counter({2016: 72, 2017: 9}) 7 Counter({2016: 181, 2017: 27}) 1 Counter({2016: 124, 2017: 15}) 11 Counter({2016: 275, 2017: 53}) 18 Counter({2016: 92, 2017: 17}) 22 Counter({2016: 78, 2017: 12}) 5 Counter({2016: 149, 2017: 30}) 16 Counter({2016: 66, 2017: 9}) 9 Counter({2016: 116, 2017: 17}) 8 Counter({2016: 124, 2017: 26}) 3 Counter({2016: 98, 2017: 18}) 2 Counter({2016: 84, 2017: 15}) 19 Counter({2016: 88, 2017: 11}) 10 Counter({2016: 144, 2017: 20}) 4 Counter({2016: 134, 2017: 15}) 17 Counter({2016: 38, 2017: 5}) 20 Counter({2016: 27, 2017: 8}) 25 Counter({2016: 150, 2017: 26}) 31 Counter({2016: 1})

‘**Daily XP70**

##### Exercise

##### Exercise

# Unique Crimes by City Block

You're in the home stretch!

Here, your data has been reshaped into a dictionary called crimes\_by\_block in which crimes are listed by city block. Your task in this exercise is to get a unique list of crimes that have occurred on a couple of the blocks that have been selected for you to learn more about. You might remember that you used set() to solve problems like this in Chapter 1.

Go for it!

##### Instructions

**100 XP**

* Create a unique list of crimes for the '001XX N STATE ST' block called n\_state\_st\_crimes and print it.
* Create a unique list of crimes for the '0000X W TERMINAL ST' block called w\_terminal\_st\_crimes and print it.
* Find the crimes committed on 001XX N STATE ST but not 0000X W TERMINAL ST. Store the result as crime\_differences and print it.

# Create a unique list of crimes for the first block: n\_state\_st\_crimes

n\_state\_st\_crimes = \_\_\_\_

# Print the list

print(n\_state\_st\_crimes)

# Create a unique list of crimes for the second block: w\_terminal\_st\_crimes

w\_terminal\_st\_crimes = \_\_\_\_

# Print the list

print(w\_terminal\_st\_crimes)

# Find the differences between the two blocks: crime\_differences

crime\_differences = \_\_\_\_

# Print the differences

print(crime\_differences)

* IPython Shell
* Slides

# Loop over the crimes\_by\_district using expansion as district and crimes

for district, crimes in crimes\_by\_district.items():

# Print the district

print(district)

# Create an empty Counter object: year\_count

year\_count = Counter()

# Loop over the crimes:

for crime in crimes:

# If there was an arrest

if crime['Arrest'] == 'true':

# Convert the Date to a datetime and get the year

year = datetime.strptime(crime['Date'], '%m/%d/%Y %I:%M:%S %p').year

# Increment the Counter for the year

year\_count[year] += 1

# Print the counter

print(year\_count)

# Create a unique list of crimes for the first block: n\_state\_st\_crimes

#print(crimes\_by\_block['001XX N STATE ST'])

n\_state\_st\_crimes = set(crimes\_by\_block['001XX N STATE ST'])

# Print the list

print(n\_state\_st\_crimes)

# Create a unique list of crimes for the second block: w\_terminal\_st\_crimes

w\_terminal\_st\_crimes = set(crimes\_by\_block['0000X W TERMINAL ST'])

# Print the list

print(w\_terminal\_st\_crimes)

# Find the differences between the two blocks: crime\_differences

crime\_differences = n\_state\_st\_crimes.difference(w\_terminal\_st\_crimes)

# Print the differences

print(crime\_differences)

# Create a unique list of crimes for the first block: n\_state\_st\_crimes

#print(crimes\_by\_block['001XX N STATE ST'])

n\_state\_st\_crimes = set(crimes\_by\_block['001XX N STATE ST'])

# Print the list

print(n\_state\_st\_crimes)

# Create a unique list of crimes for the second block: w\_terminal\_st\_crimes

w\_terminal\_st\_crimes = set(crimes\_by\_block['0000X W TERMINAL ST'])

# Print the list

print(w\_terminal\_st\_crimes)

# Find the differences between the two blocks: crime\_differences

crime\_differences = n\_state\_st\_crimes.difference(w\_terminal\_st\_crimes)

# Print the differences

print(crime\_differences)

{'DECEPTIVE PRACTICE', 'CRIMINAL TRESPASS', 'OTHER OFFENSE', 'BATTERY', 'THEFT', 'ROBBERY', 'ASSAULT', 'CRIMINAL DAMAGE'}

{'PUBLIC PEACE VIOLATION', 'DECEPTIVE PRACTICE', 'CRIMINAL TRESPASS', 'OTHER OFFENSE', 'NARCOTICS', 'THEFT', 'ASSAULT', 'CRIMINAL DAMAGE'}

{'ROBBERY', 'BATTERY'}

<script.py> output:

{'DECEPTIVE PRACTICE', 'CRIMINAL TRESPASS', 'OTHER OFFENSE', 'BATTERY', 'THEFT', 'ROBBERY', 'ASSAULT', 'CRIMINAL DAMAGE'}

{'PUBLIC PEACE VIOLATION', 'DECEPTIVE PRACTICE', 'CRIMINAL TRESPASS', 'OTHER OFFENSE', 'NARCOTICS', 'THEFT', 'ASSAULT', 'CRIMINAL DAMAGE'}

{'ROBBERY', 'BATTERY'}

## 1. Final thoughts

Congratulations, you made it! You've learned the fundamentals of data types and how to use them in many different ways with Python. You are able to bend lists, sets, and dictionaries to your will and use them to answer data science questions. You explored the collections module to maintain order, establish defaults, and count furiously. You have traveled through time with datetime objects and the Pendulum library. Then you put it all together in a case study. Again, congratulations on completing the course and thank you!

## 2. Congratulations