Dates and Times in Python
by datacamp

Python has a ‘date’ class
from datetime import date
two_hurricanes_dates = [date(2016, 10, 7), date(2017, 6, 21)]
#from biggest to smallest, year to month to day

how to print attributes
print(two_hurricanes_dates[0].year)
print(two_hurricanes_dates[0].month)
print(two_hurricanes_dates[0].day)

weekday() method
print(two_hurricanes_dates[0].weekday()
0 = Monday
1 = Tuesday
2 = Wednesday
3 = Thursday
4 = Friday
5 = Saturday
6 = Sunday

Math with dates
create two date objects
d1 = date (2017, 11, 5)
d2 = (2017, 12, 4)
create a list of the two date objects
l = [d1, d2]
example using python
print(min(l))
output 2017-11-05 (used the minimum function to print out the earliest date)

We can also subtract dates
**this gives us an object of type ‘timedelta’
delta = d2 - d1
print(delta.days)
output > 29

*you can also use timedelta from the other direction
from datetime import timedelta

```
td = timedelta(days=29)
print(d1 + td)
output > 2017-12-04
```

Side bar
Incrementing variables with +=

```
x=0                    x=0
x=x+1                  x+=1
print(x)               print(x)
output > 1             output > 1
```

**same effect, used all the time for counting

We may want to turn dates into strings
key time is when want to put dates into filenames or write dates out to CSV or Excel files

ISO 8601 format: YYYY-MM-DD

```
print( [d.isoformat()] )
output > ['2017-11-05'] gives us the date as a string
```

ISO 8601 format has other advantages
deals well with difficult dates like 2000-01-01 and 1999-12-31
*it sorts these dates appropriately but year has to be entered first

Another format strftime()
works by letting you pass a 'format string'

```
d = date(2017, 1, 5)
print(d.strftime('%Y'))
output > 2017
```

**advantage is that strftime is very flexible
can format string with more text in it

```
print(d.strftime('Year is %Y))
output > Year is 2017
```

another example

```
print(d.strftime('%Y/%m/%d))
output > 2017/01/05
```

Dealing with dates and times

```
from datetime import datetime
dt = datetime(2017, 10, 1, 15, 23, 25) #this is Oct 1, 2017 at 3:23:25p
```

you can add microseconds
in this example dt = datetime(2017, 10, 1, 15, 23, 25, 500000) # we added 500,000 microseconds

Replacing parts of a datetime
print(dt)
output 2017-10-01 15:23:25.500000
dt_hr = dt.replace(minute=0, second=0, microsecond=0)
print(dt_hr)
output 2017-10-01 15:00:00

Printing and parsing datetimes
dt = datetime(2017, 12, 30, 15, 19, 13)
print(dt.strftime("%Y-%m-%d))
this creates a string
#add in hours, minutes, seconds
print(dt.strftime("%Y-%m-%d %H:%M:%S"))
**have flexibility in how this string is formatted
random example
dt.strftime("%H:%M:%S on %Y/%m/%d"))

**official standard-compliant way to write time in computer speak
in dt.isoformat())
example output
2017-12-30T15:19:13

Parsing
parse with dt.strptime #ie string parse time
same package
from datetime import datetime
first argument is the string to turn into a datetime
second argument is the format string that says how to do it
example
dt = datetime.strptime("12/30/2017 15:19:13", "%m/%d/%Y %H:%M:%S")
print(type(dt))
output > class datetime
**need an exact match to do a string conversion

Another kind of datetime to be aware of is the Unix timestamp
many computers store datetime information this way
it is the number of seconds since January 1, 1970
considered the birth of the modern-style computer
example
ts = 1514665153.0
#convert to datetime
print(datetime.fromtimestamp(ts))

output > is the time

good example

```
# Write down the format string
fmt = "%Y-%m-%d %H:%M:%S"

# Initialize a list for holding the pairs of datetime objects
onebike_datetimes = []

# Loop over all trips
for (start, end) in onebike_datetime_strings:
  trip = {'start': datetime.strptime(start, fmt),
          'end': datetime.strptime(end, fmt)}

  # Append the trip
  onebike_datetimes.append(trip)
```

Working with durations

```
#create variables for needed datetimes
start = datetime(2017, 10, 8, 23, 46, 47)
end = datetime(2017, 10, 9, 0, 10, 57)
#subtract datetimes to create a timedelta
duration = end - start
```

**a timedelta represents what is called a duration, ie the elapsed time between events
we call method total_seconds() to get the total number of seconds of our time delta

```
print(duration.total_seconds())
```

Create a timedelta from start

```
from datetime import timedelta
delta1 = timedelta(seconds=1)
```

this makes a timedelta which corresponds to a one second duration

```
print(start)
#to get one second later
print(start + delta1)
#create another timedelta that is one day and one second
delta2 = timedelta(days=1, seconds=1)
print(start + delta2)
```

output gives us the next day and one second ahead

Can also create negative time deltas

```
delta3 = timedelta(weeks=-1)
```

```
print(start)
```
output gives one week prior

```
# Initialize a list for all the trip durations
onebike_durations = []

for trip in onebike_datetimes:
  # Create a timedelta object corresponding to the length of the trip
  trip_duration = trip['end'] - trip['start']

  # Get the total elapsed seconds in trip_duration
  trip_length_seconds = trip_duration.total_seconds()

  # Append the results to our list
  onebike_durations.append(trip_length_seconds)

# What was the total duration of all trips?
total_elapsed_time = sum(onebike_durations)

# What was the total number of trips?
number_of_trips = len(onebike_durations)

# Divide the total duration by the number of trips
print(total_elapsed_time / number_of_trips)
```

UTC offset
UTC stands for Coordinated Universal Time
the previous datetime objects that we have been working with are called “naive”
meaning they can’t be compared across different parts of the world
they are not connected to their time zone
**UTC is used when you really need to know exactly when something happened
originated in UK so time zones move west or east of that focal point
west gives you UTC - x
east gives you UTC + x

example
```
from datetime import datetime, timedelta, timezone
#our example data is from Wash DC so we are going to use ET time zone
ET = timezone(timedelta(hours=-5))
dt = datetime(2017, 12, 30, 15, 9, 3, tzinfo=ET)
print(dt)
output > 2017-12-30 15:09:03-5:00 #-5:00 will give you UTC
```

You can make a datetime “aware’ of its timezone
say we want to know what the date and time would been if the clock had been set to India Standard Time
IST = timezone(timedelta(hours=5, minutes=30))
**here we use 5.5 hours because IST is 10.5 hours ahead of Wash DC
#now we will convert to IST
print(dt.astimezone(IST))
we used the astimezone() method to ask Python to create a new datetime object corresponding to the same moment, but adjusted to a different time zone

There is a difference between adjusting timezones and changing the tzinfo directly
example
print(dt)
output > 2017-12-30 15:09:03-05:00
print(dt.replace(tzinfo=timezone.utc))
output > 2017-12-30 15:09:03+00:00
this has created a convenient object with zero UTC offset
ie clock has stayed the same but the UTC offset has shifted
**if we call the astimezone() method
print(dt.astimezone(timezone.utc))
we change both the UTC offset and the clock itself

example
```
# Import datetime, timedelta, timezone
from datetime import datetime, timedelta, timezone

# Create a timezone for Australian Eastern Daylight Time, or UTC+11
aedt = timezone(timedelta(hours=11))

# October 1, 2017 at 15:26:26, UTC+11
dt = datetime(2017, 10, 1, 15, 26, 26, tzinfo=aedt)

# Print results
print(dt.isoformat())
```

Another example
```
# Create a timezone object corresponding to UTC-4
edt = timezone(timedelta(hours=-4))

# Loop over trips, updating the start and end datetimes to be in UTC-4
for trip in onebike_datetimes[:10]:
  # Update trip['start'] and trip['end']
  trip['start'] = trip['start'].replace(tzinfo=edt)
  trip['end'] = trip['end'].replace(tzinfo=edt)
```

Another example

```
# Loop over the trips
for trip in onebike_datetimes[:10]:
  # Pull out the start
  dt = trip['start']
  # Move dt to be in UTC
  dt = dt.astimezone(timezone.utc)

  # Print the start time in UTC
  print('Original:', trip['start'], '| UTC:', dt.isoformat())
```

Time zone database

```
from datetime import datetime
from dateutil import tz
et = tz.gettz(‘America/New_York’)
```

other examples:
‘America/Mexico_City’
‘Europe/London’
‘Africa/Accra’
This is dynamic and will adjust the UTC offset depending on the date and time ie daylight savings time
example

```
#last ride
last = datetime(2017, 12, 30, 15, 9, 3, tzinfo=et)
print(last)
output > 2017-12-30 15:09:03-05:00
#first ride
first = datetime(2017, 10, 1, 15, 23, 25, tzinfo=et)
print(first)
output > 2017-10-01 15:23:25-04:00
```

**see the -4:00 and -5:00, changed automatically based off daylight savings in November

example

```
# Import tz
from dateutil import tz

# Create a timezone object for Eastern Time
et = tz.gettz('America/New_York')

# Loop over trips, updating the datetimes to be in Eastern Time
```

```
for trip in onebike_datetimes[:10]:
  # Update trip['start'] and trip['end']
  trip['start'] = trip['start'].replace(tzinfo=et)
  trip['end'] = trip['end'].replace(tzinfo=et)
```

Another example

```
# Create the timezone object
ist = tz.gettz('Asia/Kolkata')

# Pull out the start of the first trip
local = onebike_datetimes[0]['start']

# What time was it in India?
notlocal = local.astimezone(ist)

# Print them out and see the difference
print(local.isoformat())
print(notlocal.isoformat())
```

Start of Daylight Saving Time

```
from dateutil import tx
eastern = tz.gettz(‘America/New_York’)
#in EST
spring_ahead_159am = datetime(2017, 3, 12, 1, 59, 59, tzinfo = eastern)
#in EDT
spring_ahead_3am = datetime(2017, 3, 12, 3, 0, 0, tzinfo = eastern)
```

example

```
# Import datetime, timedelta, tz, timezone
from datetime import datetime, timedelta, timezone
from dateutil import tz

# Start on March 12, 2017, midnight, then add 6 hours
start = datetime(2017, 3, 12, tzinfo = tz.gettz('America/New_York'))
end = start + timedelta(hours=6)
print(start.isoformat() + " to " + end.isoformat())

# How many hours have elapsed?
print((end - start).total_seconds()/(60*60))

# What if we move to UTC?
print((end.astimezone(timezone.utc) - start.astimezone(timezone.utc))\
      .total_seconds()/(60*60))
```

Example

```
# Import datetime and tz
from datetime import datetime
from dateutil import tz

# Create starting date
dt = datetime(2000, 3, 29, tzinfo = tz.gettz('Europe/London'))

# Loop over the dates, replacing the year, and print the ISO timestamp
for y in range(2000, 2011):
  print(dt.replace(year=y).isoformat())
```

Ending Daylight Saving Time

```
eastern = tz.gettz('US/Eastern')
first_1am = datetime(2017, 11, 5, 1, 0, 0, tzinfo=eastern)
tz.datetime_ambiguous(first_1am)
output > True
```

this tells us that yes this is a time which could occur at two different UTC moments in this timezone

```
#create a second datetime with the same date and time
second_1am = datetime(2017, 11, 5, 1, 0, 0, tzinfo=eastern)
second_1am = tz.enfold(second_1am)
```

this method takes the argument of the datetime we want to mark and says this datetime belongs to the second time the wall clock struck 1am this day and not the first

```
(first_1am - second_1am).total_seconds()
output > 0.0
```

**enfold doesn't change any of the behavior of a datetime
acts as a placeholder
up to further parts of the program to pay attention and do something with it

we need to convert to UTC which is unambiguous

```
first_1am = first_1am.astimezone(tz.UTC)
second_1am = second_1am.astimezone(tz.UTC)
(second_1am - first_1am).total_seconds()
output > 3600.0
```

tells us that these two outputs are exactly an hour apart

Example

```
# Loop over trips
for trip in onebike_datetimes:
  # Rides with ambiguous start
  if tz.datetime_ambiguous(trip['start']):
    print("Ambiguous start at " + str(trip['start']))
  # Rides with ambiguous end
  if tz.datetime_ambiguous(trip['end']):
    print("Ambiguous end at " + str(trip['end']))
```

Example

```
trip_durations = []
for trip in onebike_datetimes:
  # When the start is later than the end, set the fold to be 1
  if trip['start'] > trip['end']:
    trip['end'] = tz.enfold(trip['end'])
  # Convert to UTC
  start = trip['start'].astimezone(timezone.utc)
  end = trip['end'].astimezone(timezone.utc)

  # Subtract the difference
  trip_length_seconds = (end-start).total_seconds()
  trip_durations.append(trip_length_seconds)

# Take the shortest trip duration
print("Shortest trip: " + str(min(trip_durations)))
```

Reading date and time data in Pandas
get a particular column > rides[‘Start date’]
get a particular row > rides.iloc[2] > class object

If we want pandas to treat columns as datetimes, we can use the argument parse_dates in read_csv()

```
rides = pd.read_csv(‘capital-onebike.csv’, parse_dates = [‘Start date’, ‘End date’])
```

pandas is smart in figuring out the proper datetime format but if need to fix or change

```
rides[‘Start date’] = pd.to_datetime(rides[‘Start date’], format = “%Y-%m-%d %H:%M:%S”)
```

now

```
rides[‘Start date’].iloc[2]
```

output > Timestamp class

now that they are datetimes

```
#create a duration column
rides[‘Duration’] = rides[‘End date’] - rides[’Start date’]
```

print(rides[‘Duration’].head(5))
we can convert this new column into seconds
rides[‘Duration’].dt.total_seconds().head()

Can summarize some datetime data in pandas
#average time out of the dock
rides[‘Duration’].mean()
output > timedelta object with mean time out

can use .sum()
or create percent out of dock
rides[‘Duration’].sum() / timedelta(days=91)
#this is the number of days between start and end date

#percent of rides by members
rides[‘Member type’].value_counts() / len(rides)

further examples
rides[‘Duration seconds’] = rides[‘Duration’].dt.total_seconds()
#average duration per member type
rides.groupby(‘Member type’)[‘Duration seconds’].mean()
.groupby() takes a column name and does all subsequent operations on each group

we can also group by time using the .resample() method
#average duration by month
rides.resample(‘M’, on = ’Start date’)[‘Duration seconds’].mean()
‘M’ for month

#size per group
rides.groupby(‘Member type’).size()

#first ride per group
rides.groupby(‘Member type’).first()

Plot results
rides.resample(‘M’, on = ‘Start date’)[‘Duration seconds’].mean().plot()

Example
# Create joyrides
joyrides = (rides['Start station'] == rides['End station'])

# Total number of joyrides

```
print("{} rides were joyrides".format(joyrides.sum()))

# Median of all rides
print("The median duration overall was {:.2f} seconds"\
      .format(rides['Duration'].median()))

# Median of joyrides
print("The median duration for joyrides was {:.2f} seconds"\
      .format(rides[joyrides]['Duration'].median()))

# Import matplotlib
import matplotlib.pyplot as plt

# Resample rides to monthly, take the size, plot the results
rides.resample('M', on = 'Start date')\
  .size()\
  .plot(ylim = [0, 150])

# Show the results
plt.show()

# Resample rides to be monthly on the basis of Start date
monthly_rides = rides.resample('M', on = 'Start date')['Member type']

# Take the ratio of the .value_counts() over the total number of rides
print(monthly_rides.value_counts() / monthly_rides.size())

# Group rides by member type, and resample to the month
grouped = rides.groupby('Member type')\
  .resample('M', on='Start date')

# Print the median duration for each group
print(grouped['Duration'].median())
```

Timezones in pandas
start off ‘naive’
rides[‘Duration’].dt.total_seconds().min()
output > -minutes #does not account for daylight savings

rides[’Start date’].head(3).dt.tz_localize(‘America/New_York’)
if we try to set a timezone
rides[‘Start date’] = rides[‘Start date’].dt.tz_localize(‘America/New_York’)
this will not work

we’ll get an AmbiguousTimeError
how to handle this
rides[‘Start date’] = rides[‘Start date’].dt.tz_localize(‘America/New_York’, ambiguous=‘NaT’)
rides[‘End date’] = rides[‘End date’].dt.tz_localize(‘America/New_York’, ambiguous=‘NaT’)
by passing the string ‘NaT’, we are saying if the converter gets confused, it should set the bad result as ‘Not a Time’
pandas is smart enough to skip over NaTs when it sees them, so methods like .min() will just ignore this one row

now that the timezones are fixed recalculate durations
rides[‘Duration’] = rides[‘End date’] - rides[‘Start date’]
now let’s look at the minimum
rides[‘Duration’].dt.total_seconds().min()
now intstead of getting a negative output we get a positive one

other common datetime operations in pandas
rides[‘Start date’].head(3).dt.year or dt.month or dt.day

a unique datetime operation to pandas
.dt.day_name()
gives you the day of the week for each element in a datetime Series

Can also shift the indexes forward or backward
rides[‘End date’].shift(1).head(3)
example of usefulness
lining up the end times of each row with the start time of the next one
this would allow you to compare each ride to the previous one

Example
```
# Localize the Start date column to America/New_York
rides['Start date'] = rides['Start date'].dt.tz_localize('America/New_York',
                                         ambiguous='NaT')

# Print first value
print(rides['Start date'].iloc[0])

# Convert the Start date column to Europe/London
rides['Start date'] = rides['Start date'].dt.tz_convert('Europe/London')

# Print the new value
print(rides['Start date'].iloc[0])
```

example

```
# Shift the index of the end date up one; now subract it from the start date
rides['Time since'] = rides['Start date'] - (rides['End date'].shift(1))

# Move from a timedelta to a number of seconds, which is easier to work with
rides['Time since'] = rides['Time since'].dt.total_seconds()

# Resample to the month
monthly = rides.resample('M', on='Start date')

# Print the average hours between rides each month
print(monthly['Time since'].mean()/(60*60))
```

Recap
Dates and calendars
the date() class takes a year, month, and day as arguments
a date objuect has accessors like .year, and also methods like .weekday()
date objects can be compared like numbers, using min(), max(), and sort()
you can subtract one date from another to get a timedelta
to turn date objects into strings, use the .isoformat() or .strftime() methods

Combining dates and times
the datetime() class takes all the arguments of date(), plus an hour, minute, second, and microsecond
all of the additional arguments are optional; otherwise, they're set to zero by default
you can replace any value in a datetime with .replace() method
convert a timedelta into an integer with its .total_seconds() method
turn strings into dates with .strptime() and dates into strings with .strftime()

Timezones and daylight savings
a datetime is 'timezone aware' when it has its tzinfo set, otherwise it is 'timezone naive'
setting a timezone tells a datetime how to align itself to UTC, the universal time standard
use the .replace() method to change the timezone of a datetime, leaving the date and time the same
use the .astimezone() method to shift the date and time to match the new timezone
dateutil.tz provides a comprehensive, updated timezone database

Easy and powerful timestamps in pandas

when reading a csv, set the parse_dates argument to be the list of columns which should be parsed as datetimes
if setting parse_dates doesn't work, use the pd.to_datetime() function
grouping rows with .groupby() lets you calculate aggregates per group, such as .first(), .min(), or .mean()
.resample() group rows on the basis of a datetime column, by year, month, day, and so on
use .tz_localize() to set a timezone, keeping the date and time the same
use .tz_convert() to change the date and time to match a new timezone