Learning_datetime_lib

February 15, 2018

1 Dates and Times in Python

1.1 Manipulating Time Already

Here are the datetime docs: https://docs.python.org/3/library/datetime.html
Examples

```
>>> import datetime
>>> now = datetime.datetime.now()
>>> morning = now.replace(hour=9, minute=0)
```

The above will make a variable now that represents now, and then change the time to 9am in the variable morning.

```
>>> datetime.datetime(2014, 10) - datetime.datetime(2014, 9)
```

The above will give back a datetime.timedelta object.
Importing datetime and examining methods with `dir()` :

```
In [1]: import datetime
        dir(datetime)
```

```
Out[1]: ['MAXYEAR',
         'MINYEAR',
         '__builtins__',
         '__cached__',
         '__doc__',
         '__file__',
         '__loader__',
         '__name__',
         '__package__',
         '__spec__',
         '_divide_and_round',
         'date',
         'datetime',
         'datetime_CAPI',
         'time',
         'timedelta',
         'timezone',
         'tzinfo']
```

Getting the date and time when now() method was run :

```
In [2]: datetime.datetime.now()

Out[2]: datetime.datetime(2017, 8, 22, 21, 47, 0, 633192)
```

Saving `now()` output to a variable :

```
In [3]: treehouse_start = datetime.datetime.now()
        treehouse_start

Out[3]: datetime.datetime(2017, 8, 22, 21, 47, 0, 644776)
```

Replacing the time of the previous variable with `datetime.replace()` method :

```
In [4]: treehouse_start = treehouse_start.replace(hour=9, minute=0, second=0, microsecond=0)
        treehouse_start

Out[4]: datetime.datetime(2017, 8, 22, 9, 0)
```

Setting the date and time :

```
In [5]: print('datetime.datetime(Year, Month, Day, Hour)')
        th_start = datetime.datetime(2017, 7, 17, 9)
        th_start

datetime.datetime(Year, Month, Day, Hour)


Out[5]: datetime.datetime(2017, 7, 17, 9, 0)
```

Subtracting `two` `datetimes` to get a `timedelta` :

```
In [6]: print("datetime.timedelta(Days, Seconds, Microseconds)")
        time_worked = datetime.datetime.now() - th_start
        time_worked

datetime.timedelta(Days, Seconds, Microseconds)


Out[6]: datetime.timedelta(36, 46020, 681366)
```

Grabbing specific info from a `timedelta` variable :

```
In [7]: print("Days:", time_worked.days)
        print("Seconds:", time_worked.seconds)
        print("Microseconds:", time_worked.microseconds)

Days: 36
Seconds: 46020
Microseconds: 681366
```

Looking at the methods in time_worked object :

```
In [8]: dir(time_worked)
```

```
Out[8]: ['__abs__',
         '__add__',
         '__bool__',
         '__class__',
         '__delattr__',
         '__dir__',
         '__divmod__',
         '__doc__',
         '__eq__',
         '__floordiv__',
         '__format__',
         '__ge__',
         '__getattribute__',
         '__gt__',
         '__hash__',
         '__init__',
         '__init_subclass__',
         '__le__',
         '__lt__',
         '__mod__',
         '__mul__',
         '__ne__',
         '__neg__',
         '__new__',
         '__pos__',
         '__radd__',
         '__rdivmod__',
         '__reduce__',
         '__reduce_ex__',
         '__repr__',
         '__rfloordiv__',
         '__rmod__',
         '__rmul__',
         '__rsub__',
         '__rtruediv__',
         '__setattr__',
         '__sizeof__',
         '__str__',
         '__sub__',
         '__subclasshook__',
         '__truediv__',
         'days',
         'max',
         'microseconds',
         'min',
         'resolution',
         'seconds',
         'total_seconds']
```

Calulating hours in a `timedelta` using its seconds :

```
In [9]: print("Hours Worked:", round(time_worked.seconds/3600))

Hours Worked: 13
```

1.2 Time Deltas

timedelta objects represent gaps in time. They are returned when you subtract one *datetime* from another. They can also be assigned to a variable and then used to augment *datetime* objects.

Setting the clock forward three days with a `timedelta` :

```
In [10]: now = datetime.datetime.now()
         three_days = datetime.timedelta(days=3)
         print('        Currnet Date and Time:', now)
         print('Date and Time 3 days from now:', now + three_days)

        Currnet Date and Time: 2017-08-22 21:47:00.729506
Date and Time 3 days from now: 2017-08-25 21:47:00.729506
```

Setting the clock backward five days with a `timedelta` :

```
In [11]: now = datetime.datetime.now()
         minus_5_days = datetime.timedelta(days=-5)
         print("datetime.datetime.now() + datetime.timedelta(days=-5)")
         print("                            OR")
         print("datetime.datetime.now() - datetime.timedelta(days=5)\n")
         print('        Currnet Date and Time:', now)
         print('Date and Time 3 days from now:', now + minus_5_days)

datetime.datetime.now() + datetime.timedelta(days=-5)
                            OR
datetime.datetime.now() - datetime.timedelta(days=5)

        Currnet Date and Time: 2017-08-22 21:47:00.743177
Date and Time 3 days from now: 2017-08-17 21:47:00.743177
```

Using a date without a time :

```
In [12]: now.date()

Out[12]: datetime.date(2017, 8, 22)
```

Using a time without a date :

```
In [13]: now.time()

Out[13]: datetime.time(21, 47, 0, 743177)
```

Specifying time intervals for `timedeltas` :

```
In [14]: hour = datetime.timedelta(hours=1)
         hour

Out[14]: datetime.timedelta(0, 3600)
```

Multiplying `timedeltas` with integers :

```
In [15]: hour * 9

Out[15]: datetime.timedelta(0, 32400)
```

Setting a future time and date :

```
In [16]: tomorrow = datetime.datetime.now().replace(hour=9, minute=0) + datetime.timedelta(days
         print(datetime.datetime.now())
         print(tomorrow)
         tomorrow

2017-08-22 21:47:00.796604
2017-08-23 09:00:00.796509


Out[16]: datetime.datetime(2017, 8, 23, 9, 0, 0, 796509)
```

Appointment Example :

```
In [17]: appointment = datetime.timedelta(minutes=45)
         start = datetime.datetime.now()
         end = start + appointment
         print(start)
         print(end)
         end

2017-08-22 21:47:00.809832
2017-08-22 22:32:00.809832


Out[17]: datetime.datetime(2017, 8, 22, 22, 32, 0, 809832)
```

1.3 Today and Tomorrow

Let's look at the `.now()` and `.today()` methods. * These methods use your system's local time to calculate their values.

```
In [18]: datetime.datetime.now()

Out[18]: datetime.datetime(2017, 8, 22, 21, 47, 0, 821954)

In [19]: datetime.datetime.today()

Out[19]: datetime.datetime(2017, 8, 22, 21, 47, 0, 831869)
```

Both outputs you'll notice are almost exactly the same, why is this, why have two commands that do the exact same thing? Well, `.now()` takes a timezone argument, which allows you to specify where the now is.

Combining Dates and times : * `datetime.time()` as is, without any arguments, produces a time of midnight.

```
In [20]: datetime.datetime.combine(datetime.date.today(), datetime.time())

Out[20]: datetime.datetime(2017, 8, 22, 0, 0)
```

Grabbing elements from a datetime object :

```
In [21]: today = datetime.datetime.today()
         today

Out[21]: datetime.datetime(2017, 8, 22, 21, 47, 0, 853511)

In [22]: today.hour

Out[22]: 21

In [23]: today.year

Out[23]: 2017

In [24]: today.month

Out[24]: 8

In [25]: today.minute

Out[25]: 47

In [26]: today.weekday() # Python's weeks start on Monday which is zero

Out[26]: 1

In [27]: today.timestamp()  # Posix Timestamp: seconds since Jan 1, 1970
```

```
Out[27]: 1503463620.853511
```

Code Challenge : Write a function named minutes that takes two datetimes and, using timedelta.total_seconds() to get the number of seconds, returns the number of minutes, rounded, between them. The first will always be older and the second newer. You'll need to subtract the first from the second.

```
In [28]: import datetime

         def minutes(dt1, dt2):
             timedelta = dt2-dt1
             return round(timedelta.total_seconds() / 60)
```

2 Dating Methods

New Terms * `strftime` - Method to create a string from a datetime * `strptime` - Method to create a datetime from a string according to a format string * Guide: https://docs.python.org/3/library/datetime.html?highlight=datetime#strftime-and-strptime-behavior

- `strftime` is str from time
- `strptime` is str parsed to time

```
In [29]: today
```

```
Out[29]: datetime.datetime(2017, 8, 22, 21, 47, 0, 853511)
```

`strftime` Dating Method

```
In [ ]: today.strftime('%m/%d/%Y')
```

`strptime` Dating Method

```
In [31]: datetime.datetime.strptime('2017-08-21', '%Y-%m-%d')
```

```
Out[31]: datetime.datetime(2017, 8, 21, 0, 0)
```

`strptime` with both a date and time

```
In [32]: datetime.datetime.strptime('2017-08-21 10:00', '%Y-%m-%d %H:%M')
```

```
Out[32]: datetime.datetime(2017, 8, 21, 10, 0)
```

Code Challenge :
Challenge Task 1 of 2
Create a function named to_string that takes a datetime and gives back a string in the format "24 September 2012".

```
In [33]: ## Examples
         # to_string(datetime_object) => "24 September 2012"
         # from_string("09/24/12 18:30", "%m/%d/%y %H:%M") => datetime

         def to_string(dt):
             return dt.strftime('%d %B %Y')
```

Challenge Task 2 of 2
Create a new function named from_string that takes two arguments: a date as a string and an strftime-compatible format string, and returns a datetime created from them.

```
In [34]: def from_string(date, dt_format):
             return datetime.datetime.strptime(date, dt_format)
```

3 Wikipedia Links App HW

```
In [35]: import datetime


         answer_format = '%m/%d'
         link_format = '%b_%d'
         link = 'https://en.wikipedia.org/wiki/{}'

         while True:
             answer = input("What date would you like? Please use the MM/DD format. Enter 'qui
             if answer.upper() == 'QUIT':
                 break

             try:
                 date = datetime.datetime.strptime(answer, answer_format)
                 output = link.format(date.strftime(link_format))
                 print(output)
             except ValueError:
                 print("That's not a valid date. Please try again.")
```

```
What date would you like? Please use the MM/DD format. Enter 'quit' to quit. quit
```

Code Challenge :
Write a function named time_tango that takes a date and a time. It should combine them into a datetime and return it.

```
In [36]: import datetime

         def time_tango(date, time):
             return datetime.datetime.combine(date, time)
```

4 The Question Classes & the Plan

Building an Addition and Multiplication Quiz app

```
In [37]: #questions.py

         class Question:
             answer = None
             text = None


         class Add(Question):
             def __init__(self, num1, num2):
                 self.text = '{} + {}'.format(num1, num2)
                 self.answer = num1 + num2


         class Multiply(Question):
             def __init__(self, num1, num2):
                 self.text = '{} X {}'.format(num1, num2)
                 self.answer = num1 * num2
```

```
In [38]: #from Question import Add

         add1 = Add(5,7)
         add1.answer
```

```
Out[38]: 12
```

```
In [39]: add1.text
```

```
Out[39]: '5 + 7'
```

```
In [40]: # quiz.py
         import datetime
         import random

         #from questions import Add, Multiply


         class Quiz:
             questions = []
             answers = []


             def __init__(self):
                 question_types = (Add, Multiply)
                 #question_types[0](1,5)
```

```
                for _ in range(10):
                    num1 = random.randint(1,10)
                    num2 = random.randint(1,10)
                    question = random.choice(question_types)(num1, num2)
                    self.questions.append(question)

            def take_quiz(self):
                self.start_time = datetime.datetime.now()
                for question in self.questions:
                    self.answers.append(self.ask(question))
                else:
                    self.end_time = datetime.datetime.now()
                return self.summary()

            def ask(self, question):
                correct = False
                question_start = datetime.datetime.now()
                answer = input(question.text + ' = ')
                if answer == str(question.answer):
                    correct = True
                question_end = datetime.datetime.now()
                return correct, question_end - question_start

            def total_correct(self):
                total = 0
                for answer in self.answers:
                    if answer[0]:
                        total += 1
                return total

            def summary(self):
                print('You got {} out of {} correct'.format(
                    self.total_correct(), len(self.questions)))
                print("It took {} seconds total".format((
                    self.end_time-self.start_time).seconds))
        # Quiz().take_quiz()
```

```
In [41]: # quiz1 = Quiz()
         # quiz1.answers
```

```
In [42]: # quiz1.questions
```

```
In [43]: # quiz1.questions[0].text
```

```
In [44]: # quiz1.questions[0].answer
```

```
In [45]: #Quiz().take_quiz()
```

Code Challenge :

Write a function named delorean that takes an integer. Return a datetime that is that many hours ahead from starter.

```
In [46]: import datetime

         starter = datetime.datetime(2015, 10, 21, 16, 29)

         def delorean(integer):
             return starter + datetime.timedelta(hours=integer)
```

```
In [47]: delorean(5)
```

```
Out[47]: datetime.datetime(2015, 10, 21, 21, 29)
```

Code Challenge :

Write a function named time_machine that takes an integer and a string of “minutes”, “hours”, “days”, or “years”. This describes a timedelta. Return a datetime that is the timedelta’s duration from the starter datetime.

```
In [48]: import datetime

         starter = datetime.datetime(2015, 10, 21, 16, 29)

         # Remember, you can't set "years" on a timedelta!
         # Consider a year to be 365 days.

         ## Example
         # time_machine(5, "minutes") => datetime(2015, 10, 21, 16, 34)

         def time_machine(integer, time_str):
             if time_str == 'minutes':
                 return starter + datetime.timedelta(minutes=integer)
             elif time_str == 'hours':
                 return starter + datetime.timedelta(hours=integer)
             elif time_str == 'days':
                 return starter + datetime.timedelta(days=integer)
             else:
                 return starter + datetime.timedelta(days=integer*365)
```

```
In [49]: time_machine(5,'years')
```

```
Out[49]: datetime.datetime(2020, 10, 19, 16, 29)
```

Code Challenge :

Create a function named timestamp_oldest that takes any number of POSIX timestamp arguments. Return the oldest one as a datetime object.

Remember, POSIX timestamps are floats and lists have a .sort() method.

```
In [50]: # If you need help, look up datetime.datetime.fromtimestamp()
         # Also, remember that you *will not* know how many timestamps
         # are coming in.
         import datetime

         def timestamp_oldest(*posix_args):
             return datetime.datetime.fromtimestamp(min(*posix_args))
```

5 Introduction to Timezones

Videos * The Problem with Time & Timezones - from Computerphile https://www.youtube.com/watch?v=-5wpm-gesOY * Strangest Timezones of the World - from WonderWhy https://www.youtube.com/watch?v=uW6QqcmCfm8

New Terms

- **timezone** - datetime type that holds an offset from UTC and allows us to move a datetime around the world
- **astimezone** - method for converting an aware datetime to another timezone

```
In [51]: pacific = datetime.timezone(datetime.timedelta(hours=-8))
         eastern = datetime.timezone(datetime.timedelta(hours=-5))

In [52]: naive = datetime.datetime(2017, 4, 21, 9)
         naive

Out[52]: datetime.datetime(2017, 4, 21, 9, 0)

In [53]: aware = datetime.datetime(2017, 4, 21, 9, tzinfo=pacific)
         aware

Out[53]: datetime.datetime(2017, 4, 21, 9, 0, tzinfo=datetime.timezone(datetime.timedelta(-1, 5

In [54]: naive.astimezone()

Out[54]: datetime.datetime(2017, 4, 21, 9, 0, tzinfo=datetime.timezone(datetime.timedelta(-1, 6

In [55]: aware.astimezone(eastern)

Out[55]: datetime.datetime(2017, 4, 21, 12, 0, tzinfo=datetime.timezone(datetime.timedelta(-1,

In [56]: auckland = datetime.timezone(datetime.timedelta(hours=13))

In [57]: aware.astimezone(auckland)

Out[57]: datetime.datetime(2017, 4, 22, 6, 0, tzinfo=datetime.timezone(datetime.timedelta(0, 46

In [58]: mumbai = datetime.timezone(datetime.timedelta(hours=5, minutes=30))

In [59]: aware.astimezone(mumbai)
```

```
Out[59]: datetime.datetime(2017, 4, 21, 22, 30, tzinfo=datetime.timezone(datetime.timedelta(0,
```

Code Challenge :
Challenge Task 1 of 3
Create a variable named moscow that holds a datetime.timezone object at +4 hours.

```
In [60]: moscow = datetime.timezone(datetime.timedelta(hours=4))
```

Challenge Task 2 of 3
Now create a timezone variable named pacific that holds a timezone at UTC-08:00.

```
In [61]: pacific = datetime.timezone(datetime.timedelta(hours=-8))
```

Challenge Task 3 of 3
Finally, make a third variable named india that hold's a timezone at UTC+05:30.

```
In [62]: india = datetime.timezone(datetime.timedelta(hours=5, minutes=30))
```

Code Challenge :
Challenge Task 1 of 2
naive is a datetime with no timezone. Create a new timezone for US/Pacific, which is 8 hours behind UTC (UTC-08:00). Then make a new variable named hill_valley that is naive with its tzinfo attribute replaced with the US/Pacific timezone you made.

```
In [63]: import datetime

         naive = datetime.datetime(2015, 10, 21, 4, 29)
         pacific = datetime.timezone(datetime.timedelta(hours=-8))
         hill_valley = naive.replace(tzinfo=pacific)
```

Challenge Task 2 of 2
Great, but replace just sets the timezone, it doesn't move the datetime to the new timezone. Let's move one. Make a new timezone that is UTC+01:00. Create a new variable named paris that uses your new timezone and the astimezone method to change hill_valley to the new timezone.

```
In [64]: paris = hill_valley.astimezone(datetime.timezone(datetime.timedelta(hours=+1)))
```

6 Actually, Use pytz Instead

Format string

- fmt = '%Y-%m-%d %H:%M:%S %Z%z' * Z = Name of Timezone * z = offset of timezone from UTC

Links

- pytz docs http://pythonhosted.org//pytz/
- More about pytz from SaltyCrane http://www.saltycrane.com/blog/2009/05/converting-time-zones-datetime-objects-python/

Libraries

- Chronyk https://pypi.python.org/pypi/Chronyk/0.9.1
- delorean http://delorean.readthedocs.org/en/latest/

```
In [65]: import pytz
```

```
In [66]: pacific = pytz.timezone('US/Pacific')
         eastern = pytz.timezone('US/Eastern')
         fmt = '%Y-%m-%d %H:%M:%S %Z%z'
         utc = pytz.utc
```

`.localize` is used for naive datetimes.

```
In [67]: start = pacific.localize(datetime.datetime(2014, 4, 21, 9))
         start.strftime(fmt)

Out[67]: '2014-04-21 09:00:00 PDT-0700'
```

`.astimezone` is used for aware datetimes.

```
In [68]: start_eastern = start.astimezone(eastern)
         start_eastern

Out[68]: datetime.datetime(2014, 4, 21, 12, 0, tzinfo=<DstTzInfo 'US/Eastern' EDT-1 day, 20:00
```

Creating a datetime with the UTC timezone

```
In [69]: start_utc = datetime.datetime(2014, 4, 21, 12, 0, tzinfo=utc)
         start_utc.strftime(fmt)

Out[69]: '2014-04-21 12:00:00 UTC+0000'
```

Switching to Pacific Timezone

```
In [70]: start_pacific = start_utc.astimezone(pacific)
```

Creating Timezones with pytz

```
In [71]: auckland = pytz.timezone('Pacific/Auckland')
         mumbai = pytz.timezone('Asia/Calcutta')
```

Apollo 13 Launch Time Examples * Create naive datetime of launch time

```
In [72]: apollo_13_naive = datetime.datetime(1970, 4, 11, 14, 13)
```

Add eastern timezone to naive launch time

```
In [73]: apollo_13_eastern = eastern.localize(apollo_13_naive)
         apollo_13_eastern

Out[73]: datetime.datetime(1970, 4, 11, 14, 13, tzinfo=<DstTzInfo 'US/Eastern' EST-1 day, 19:0
```

Convert eastern launch time to utc launch time * This allows for easier conversion to all other timezones, because UTC does not have Day Light Savings Time.

```
In [74]: apollo_13_utc = apollo_13_eastern.astimezone(utc)
         apollo_13_utc
```

```
Out[74]: datetime.datetime(1970, 4, 11, 19, 13, tzinfo=<UTC>)
```

```
In [75]: apollo_13_utc.astimezone(pacific).strftime(fmt)
```

```
Out[75]: '1970-04-11 11:13:00 PST-0800'
```

```
In [76]: apollo_13_utc.astimezone(auckland).strftime(fmt)
```

```
Out[76]: '1970-04-12 07:13:00 NZST+1200'
```

```
In [77]: apollo_13_utc.astimezone(mumbai).strftime(fmt)
```

```
Out[77]: '1970-04-12 00:43:00 IST+0530'
```

Two handy methods for finding timezones

```
In [78]: pytz.all_timezones
```

```
Out[78]: ['Africa/Abidjan',
          'Africa/Accra',
          'Africa/Addis_Ababa',
          'Africa/Algiers',
          'Africa/Asmara',
          'Africa/Asmera',
          'Africa/Bamako',
          'Africa/Bangui',
          'Africa/Banjul',
          'Africa/Bissau',
          'Africa/Blantyre',
          'Africa/Brazzaville',
          'Africa/Bujumbura',
          'Africa/Cairo',
          'Africa/Casablanca',
          'Africa/Ceuta',
          'Africa/Conakry',
          'Africa/Dakar',
          'Africa/Dar_es_Salaam',
          'Africa/Djibouti',
          'Africa/Douala',
          'Africa/El_Aaiun',
          'Africa/Freetown',
          'Africa/Gaborone',
          'Africa/Harare',
          'Africa/Johannesburg',
          'Africa/Juba',
          'Africa/Kampala',
          'Africa/Khartoum',
          'Africa/Kigali',
          'Africa/Kinshasa',
          'Africa/Lagos',
          'Africa/Libreville',
          'Africa/Lome',
          'Africa/Luanda',
          'Africa/Lubumbashi',
          'Africa/Lusaka',
          'Africa/Malabo',
          'Africa/Maputo',
          'Africa/Maseru',
          'Africa/Mbabane',
          'Africa/Mogadishu',
          'Africa/Monrovia',
          'Africa/Nairobi',
          'Africa/Ndjamena',
          'Africa/Niamey',
          'Africa/Nouakchott',
          'Africa/Ouagadougou',
          'Africa/Porto-Novo',
          'Africa/Sao_Tome',
          'Africa/Timbuktu',
          'Africa/Tripoli',
          'Africa/Tunis',
          'Africa/Windhoek',
          'America/Adak',
          'America/Anchorage',
          'America/Anguilla',
          'America/Antigua',
          'America/Araguaina',
          'America/Argentina/Buenos_Aires',
          'America/Argentina/Catamarca',
          'America/Argentina/ComodRivadavia',
          'America/Argentina/Cordoba',
          'America/Argentina/Jujuy',
          'America/Argentina/La_Rioja',
          'America/Argentina/Mendoza',
          'America/Argentina/Rio_Gallegos',
          'America/Argentina/Salta',
          'America/Argentina/San_Juan',
          'America/Argentina/San_Luis',
          'America/Argentina/Tucuman',
          'America/Argentina/Ushuaia',
          'America/Aruba',
          'America/Asuncion',
          'America/Atikokan',
          'America/Atka',
          'America/Bahia',
          'America/Bahia_Banderas',
          'America/Barbados',
          'America/Belem',
          'America/Belize',
          'America/Blanc-Sablon',
          'America/Boa_Vista',
          'America/Bogota',
          'America/Boise',
          'America/Buenos_Aires',
          'America/Cambridge_Bay',
          'America/Campo_Grande',
          'America/Cancun',
          'America/Caracas',
          'America/Catamarca',
          'America/Cayenne',
          'America/Cayman',
          'America/Chicago',
          'America/Chihuahua',
          'America/Coral_Harbour',
          'America/Cordoba',
          'America/Costa_Rica',
          'America/Creston',
          'America/Cuiaba',
          'America/Curacao',
          'America/Danmarkshavn',
          'America/Dawson',
          'America/Dawson_Creek',
          'America/Denver',
          'America/Detroit',
          'America/Dominica',
          'America/Edmonton',
          'America/Eirunepe',
          'America/El_Salvador',
          'America/Ensenada',
          'America/Fort_Nelson',
          'America/Fort_Wayne',
          'America/Fortaleza',
          'America/Glace_Bay',
          'America/Godthab',
          'America/Goose_Bay',
          'America/Grand_Turk',
          'America/Grenada',
          'America/Guadeloupe',
          'America/Guatemala',
          'America/Guayaquil',
          'America/Guyana',
          'America/Halifax',
          'America/Havana',
          'America/Hermosillo',
          'America/Indiana/Indianapolis',
          'America/Indiana/Knox',
          'America/Indiana/Marengo',
          'America/Indiana/Petersburg',
          'America/Indiana/Tell_City',
          'America/Indiana/Vevay',
          'America/Indiana/Vincennes',
          'America/Indiana/Winamac',
          'America/Indianapolis',
          'America/Inuvik',
          'America/Iqaluit',
          'America/Jamaica',
          'America/Jujuy',
          'America/Juneau',
          'America/Kentucky/Louisville',
          'America/Kentucky/Monticello',
          'America/Knox_IN',
          'America/Kralendijk',
          'America/La_Paz',
          'America/Lima',
          'America/Los_Angeles',
          'America/Louisville',
          'America/Lower_Princes',
          'America/Maceio',
          'America/Managua',
          'America/Manaus',
          'America/Marigot',
          'America/Martinique',
          'America/Matamoros',
          'America/Mazatlan',
          'America/Mendoza',
          'America/Menominee',
          'America/Merida',
          'America/Metlakatla',
          'America/Mexico_City',
          'America/Miquelon',
          'America/Moncton',
          'America/Monterrey',
          'America/Montevideo',
          'America/Montreal',
          'America/Montserrat',
          'America/Nassau',
          'America/New_York',
          'America/Nipigon',
          'America/Nome',
          'America/Noronha',
          'America/North_Dakota/Beulah',
          'America/North_Dakota/Center',
          'America/North_Dakota/New_Salem',
          'America/Ojinaga',
          'America/Panama',
          'America/Pangnirtung',
          'America/Paramaribo',
          'America/Phoenix',
          'America/Port-au-Prince',
          'America/Port_of_Spain',
          'America/Porto_Acre',
          'America/Porto_Velho',
          'America/Puerto_Rico',
          'America/Punta_Arenas',
          'America/Rainy_River',
          'America/Rankin_Inlet',
          'America/Recife',
          'America/Regina',
          'America/Resolute',
          'America/Rio_Branco',
          'America/Rosario',
          'America/Santa_Isabel',
          'America/Santarem',
          'America/Santiago',
          'America/Santo_Domingo',
          'America/Sao_Paulo',
          'America/Scoresbysund',
          'America/Shiprock',
          'America/Sitka',
          'America/St_Barthelemy',
          'America/St_Johns',
          'America/St_Kitts',
          'America/St_Lucia',
          'America/St_Thomas',
          'America/St_Vincent',
          'America/Swift_Current',
          'America/Tegucigalpa',
          'America/Thule',
          'America/Thunder_Bay',
          'America/Tijuana',
          'America/Toronto',
          'America/Tortola',
          'America/Vancouver',
          'America/Virgin',
          'America/Whitehorse',
          'America/Winnipeg',
          'America/Yakutat',
          'America/Yellowknife',
          'Antarctica/Casey',
          'Antarctica/Davis',
          'Antarctica/DumontDUrville',
          'Antarctica/Macquarie',
          'Antarctica/Mawson',
          'Antarctica/McMurdo',
          'Antarctica/Palmer',
          'Antarctica/Rothera',
          'Antarctica/South_Pole',
          'Antarctica/Syowa',
          'Antarctica/Troll',
          'Antarctica/Vostok',
          'Arctic/Longyearbyen',
          'Asia/Aden',
          'Asia/Almaty',
          'Asia/Amman',
          'Asia/Anadyr',
          'Asia/Aqtau',
          'Asia/Aqtobe',
          'Asia/Ashgabat',
          'Asia/Ashkhabad',
          'Asia/Atyrau',
          'Asia/Baghdad',
          'Asia/Bahrain',
          'Asia/Baku',
          'Asia/Bangkok',
          'Asia/Barnaul',
          'Asia/Beirut',
          'Asia/Bishkek',
          'Asia/Brunei',
          'Asia/Calcutta',
          'Asia/Chita',
          'Asia/Choibalsan',
          'Asia/Chongqing',
          'Asia/Chungking',
          'Asia/Colombo',
          'Asia/Dacca',
          'Asia/Damascus',
          'Asia/Dhaka',
          'Asia/Dili',
          'Asia/Dubai',
          'Asia/Dushanbe',
          'Asia/Famagusta',
          'Asia/Gaza',
          'Asia/Harbin',
          'Asia/Hebron',
          'Asia/Ho_Chi_Minh',
          'Asia/Hong_Kong',
          'Asia/Hovd',
          'Asia/Irkutsk',
          'Asia/Istanbul',
          'Asia/Jakarta',
          'Asia/Jayapura',
          'Asia/Jerusalem',
          'Asia/Kabul',
          'Asia/Kamchatka',
          'Asia/Karachi',
          'Asia/Kashgar',
          'Asia/Kathmandu',
          'Asia/Katmandu',
          'Asia/Khandyga',
          'Asia/Kolkata',
          'Asia/Krasnoyarsk',
          'Asia/Kuala_Lumpur',
          'Asia/Kuching',
          'Asia/Kuwait',
          'Asia/Macao',
          'Asia/Macau',
          'Asia/Magadan',
          'Asia/Makassar',
          'Asia/Manila',
          'Asia/Muscat',
          'Asia/Nicosia',
          'Asia/Novokuznetsk',
          'Asia/Novosibirsk',
          'Asia/Omsk',
          'Asia/Oral',
          'Asia/Phnom_Penh',
          'Asia/Pontianak',
          'Asia/Pyongyang',
          'Asia/Qatar',
          'Asia/Qyzylorda',
          'Asia/Rangoon',
          'Asia/Riyadh',
          'Asia/Saigon',
          'Asia/Sakhalin',
          'Asia/Samarkand',
          'Asia/Seoul',
          'Asia/Shanghai',
          'Asia/Singapore',
          'Asia/Srednekolymsk',
          'Asia/Taipei',
          'Asia/Tashkent',
          'Asia/Tbilisi',
          'Asia/Tehran',
          'Asia/Tel_Aviv',
          'Asia/Thimbu',
          'Asia/Thimphu',
          'Asia/Tokyo',
          'Asia/Tomsk',
          'Asia/Ujung_Pandang',
          'Asia/Ulaanbaatar',
          'Asia/Ulan_Bator',
          'Asia/Urumqi',
          'Asia/Ust-Nera',
          'Asia/Vientiane',
          'Asia/Vladivostok',
          'Asia/Yakutsk',
          'Asia/Yangon',
          'Asia/Yekaterinburg',
          'Asia/Yerevan',
          'Atlantic/Azores',
          'Atlantic/Bermuda',
          'Atlantic/Canary',
          'Atlantic/Cape_Verde',
          'Atlantic/Faeroe',
          'Atlantic/Faroe',
          'Atlantic/Jan_Mayen',
          'Atlantic/Madeira',
          'Atlantic/Reykjavik',
          'Atlantic/South_Georgia',
          'Atlantic/St_Helena',
          'Atlantic/Stanley',
          'Australia/ACT',
          'Australia/Adelaide',
          'Australia/Brisbane',
          'Australia/Broken_Hill',
          'Australia/Canberra',
          'Australia/Currie',
          'Australia/Darwin',
          'Australia/Eucla',
          'Australia/Hobart',
          'Australia/LHI',
          'Australia/Lindeman',
          'Australia/Lord_Howe',
          'Australia/Melbourne',
          'Australia/NSW',
          'Australia/North',
          'Australia/Perth',
          'Australia/Queensland',
          'Australia/South',
          'Australia/Sydney',
          'Australia/Tasmania',
          'Australia/Victoria',
          'Australia/West',
          'Australia/Yancowinna',
          'Brazil/Acre',
          'Brazil/DeNoronha',
          'Brazil/East',
          'Brazil/West',
          'CET',
          'CST6CDT',
          'Canada/Atlantic',
          'Canada/Central',
          'Canada/East-Saskatchewan',
          'Canada/Eastern',
          'Canada/Mountain',
          'Canada/Newfoundland',
          'Canada/Pacific',
          'Canada/Saskatchewan',
          'Canada/Yukon',
          'Chile/Continental',
          'Chile/EasterIsland',
          'Cuba',
          'EET',
          'EST',
          'EST5EDT',
          'Egypt',
          'Eire',
          'Etc/GMT',
          'Etc/GMT+0',
          'Etc/GMT+1',
          'Etc/GMT+10',
          'Etc/GMT+11',
          'Etc/GMT+12',
          'Etc/GMT+2',
          'Etc/GMT+3',
          'Etc/GMT+4',
          'Etc/GMT+5',
          'Etc/GMT+6',
          'Etc/GMT+7',
          'Etc/GMT+8',
          'Etc/GMT+9',
          'Etc/GMT-0',
          'Etc/GMT-1',
          'Etc/GMT-10',
          'Etc/GMT-11',
          'Etc/GMT-12',
          'Etc/GMT-13',
          'Etc/GMT-14',
          'Etc/GMT-2',
          'Etc/GMT-3',
          'Etc/GMT-4',
          'Etc/GMT-5',
          'Etc/GMT-6',
          'Etc/GMT-7',
          'Etc/GMT-8',
          'Etc/GMT-9',
          'Etc/GMT0',
          'Etc/Greenwich',
          'Etc/UCT',
          'Etc/UTC',
          'Etc/Universal',
          'Etc/Zulu',
          'Europe/Amsterdam',
          'Europe/Andorra',
          'Europe/Astrakhan',
          'Europe/Athens',
          'Europe/Belfast',
          'Europe/Belgrade',
          'Europe/Berlin',
          'Europe/Bratislava',
          'Europe/Brussels',
          'Europe/Bucharest',
          'Europe/Budapest',
          'Europe/Busingen',
          'Europe/Chisinau',
          'Europe/Copenhagen',
          'Europe/Dublin',
          'Europe/Gibraltar',
          'Europe/Guernsey',
          'Europe/Helsinki',
          'Europe/Isle_of_Man',
          'Europe/Istanbul',
          'Europe/Jersey',
          'Europe/Kaliningrad',
          'Europe/Kiev',
          'Europe/Kirov',
          'Europe/Lisbon',
          'Europe/Ljubljana',
          'Europe/London',
          'Europe/Luxembourg',
          'Europe/Madrid',
          'Europe/Malta',
          'Europe/Mariehamn',
          'Europe/Minsk',
          'Europe/Monaco',
          'Europe/Moscow',
          'Europe/Nicosia',
          'Europe/Oslo',
          'Europe/Paris',
          'Europe/Podgorica',
          'Europe/Prague',
          'Europe/Riga',
          'Europe/Rome',
          'Europe/Samara',
          'Europe/San_Marino',
          'Europe/Sarajevo',
          'Europe/Saratov',
          'Europe/Simferopol',
          'Europe/Skopje',
          'Europe/Sofia',
          'Europe/Stockholm',
          'Europe/Tallinn',
          'Europe/Tirane',
          'Europe/Tiraspol',
          'Europe/Ulyanovsk',
          'Europe/Uzhgorod',
          'Europe/Vaduz',
          'Europe/Vatican',
          'Europe/Vienna',
          'Europe/Vilnius',
          'Europe/Volgograd',
          'Europe/Warsaw',
          'Europe/Zagreb',
          'Europe/Zaporozhye',
          'Europe/Zurich',
          'GB',
          'GB-Eire',
          'GMT',
          'GMT+0',
          'GMT-0',
          'GMT0',
          'Greenwich',
          'HST',
          'Hongkong',
          'Iceland',
          'Indian/Antananarivo',
          'Indian/Chagos',
          'Indian/Christmas',
          'Indian/Cocos',
          'Indian/Comoro',
          'Indian/Kerguelen',
          'Indian/Mahe',
          'Indian/Maldives',
          'Indian/Mauritius',
          'Indian/Mayotte',
          'Indian/Reunion',
          'Iran',
          'Israel',
          'Jamaica',
          'Japan',
          'Kwajalein',
          'Libya',
          'MET',
          'MST',
          'MST7MDT',
          'Mexico/BajaNorte',
          'Mexico/BajaSur',
          'Mexico/General',
          'NZ',
          'NZ-CHAT',
          'Navajo',
          'PRC',
          'PST8PDT',
          'Pacific/Apia',
          'Pacific/Auckland',
          'Pacific/Bougainville',
          'Pacific/Chatham',
          'Pacific/Chuuk',
          'Pacific/Easter',
          'Pacific/Efate',
          'Pacific/Enderbury',
          'Pacific/Fakaofo',
          'Pacific/Fiji',
          'Pacific/Funafuti',
          'Pacific/Galapagos',
          'Pacific/Gambier',
          'Pacific/Guadalcanal',
          'Pacific/Guam',
          'Pacific/Honolulu',
          'Pacific/Johnston',
          'Pacific/Kiritimati',
          'Pacific/Kosrae',
          'Pacific/Kwajalein',
          'Pacific/Majuro',
          'Pacific/Marquesas',
          'Pacific/Midway',
          'Pacific/Nauru',
          'Pacific/Niue',
          'Pacific/Norfolk',
          'Pacific/Noumea',
          'Pacific/Pago_Pago',
          'Pacific/Palau',
          'Pacific/Pitcairn',
          'Pacific/Pohnpei',
          'Pacific/Ponape',
          'Pacific/Port_Moresby',
          'Pacific/Rarotonga',
          'Pacific/Saipan',
          'Pacific/Samoa',
          'Pacific/Tahiti',
          'Pacific/Tarawa',
          'Pacific/Tongatapu',
          'Pacific/Truk',
          'Pacific/Wake',
          'Pacific/Wallis',
          'Pacific/Yap',
          'Poland',
          'Portugal',
          'ROC',
          'ROK',
          'Singapore',
          'Turkey',
          'UCT',
          'US/Alaska',
          'US/Aleutian',
          'US/Arizona',
          'US/Central',
          'US/East-Indiana',
          'US/Eastern',
          'US/Hawaii',
          'US/Indiana-Starke',
          'US/Michigan',
          'US/Mountain',
          'US/Pacific',
          'US/Pacific-New',
          'US/Samoa',
          'UTC',
          'Universal',
          'W-SU',
          'WET',
          'Zulu']
```

```
In [79]: pytz.country_timezones['us']
```

```
Out[79]: ['America/New_York',
          'America/Detroit',
          'America/Kentucky/Louisville',
          'America/Kentucky/Monticello',
          'America/Indiana/Indianapolis',
          'America/Indiana/Vincennes',
          'America/Indiana/Winamac',
          'America/Indiana/Marengo',
          'America/Indiana/Petersburg',
          'America/Indiana/Vevay',
          'America/Chicago',
          'America/Indiana/Tell_City',
          'America/Indiana/Knox',
          'America/Menominee',
          'America/North_Dakota/Center',
          'America/North_Dakota/New_Salem',
          'America/North_Dakota/Beulah',
          'America/Denver',
          'America/Boise',
          'America/Phoenix',
          'America/Los_Angeles',
          'America/Anchorage',
          'America/Juneau',
          'America/Sitka',
          'America/Metlakatla',
          'America/Yakutat',
          'America/Nome',
          'America/Adak',
          'Pacific/Honolulu']
```

```
In [80]: pytz.country_timezones['ca'] # Canada
```

```
Out[80]: ['America/St_Johns',
          'America/Halifax',
          'America/Glace_Bay',
          'America/Moncton',
          'America/Goose_Bay',
          'America/Blanc-Sablon',
          'America/Toronto',
          'America/Nipigon',
          'America/Thunder_Bay',
          'America/Iqaluit',
          'America/Pangnirtung',
          'America/Atikokan',
          'America/Winnipeg',
          'America/Rainy_River',
          'America/Resolute',
          'America/Rankin_Inlet',
          'America/Regina',
          'America/Swift_Current',
          'America/Edmonton',
          'America/Cambridge_Bay',
          'America/Yellowknife',
          'America/Inuvik',
          'America/Creston',
          'America/Dawson_Creek',
          'America/Fort_Nelson',
          'America/Vancouver',
          'America/Whitehorse',
          'America/Dawson']
```

Code Challenge :
Challenge Task 1 of 2
starter is a naive datetime. Use pytz to make it a "US/Pacific" datetime instead and assign this converted datetime to the variable local.

```
In [81]: import datetime

         fmt = '%m-%d %H:%M %Z%z'
         starter = datetime.datetime(2015, 10, 21, 4, 29)

         local = pytz.timezone('US/Pacific').localize(starter)
         local

Out[81]: datetime.datetime(2015, 10, 21, 4, 29, tzinfo=<DstTzInfo 'US/Pacific' PDT-1 day, 17:0
```

Challenge Task 2 of 2
Now create a variable named pytz_string by using strftime with the local datetime. Use the fmt string for the formatting.

```
In [ ]: pytz_string = local.strftime(fmt)
        pytz_string

Out[ ]: '10-21 04:29 PDT-0700'
```

7 Timezonapalooza

Timezone script homework * User provides a date and time and script spits out that date time in 6 other timezones.

```
In [ ]: # meeting.py

        from datetime import datetime

        import pytz


        OTHER_TIMEZONES = [
            pytz.timezone('US/Mountain'),
            pytz.timezone('US/Central'),
            pytz.timezone('US/Eastern'),
            pytz.timezone('UTC'),
            pytz.timezone('Asia/Hong_Kong'),
            pytz.timezone('Pacific/Honolulu')
        ]

        fmt = '%Y-%m-%d %H:%M %Z%z'

        while True:
            date_input = input('When is your meeting? Please use MM/DD/YYYY HH:MM format. ')
            try:
                local_date = datetime.strptime(date_input, '%m/%d/%Y %H:%M')
            except ValueError:
                print("{} doesn't appear to be a valid date & time.".format(date_input))
            else:
                local_date = pytz.timezone('US/Pacific').localize(local_date)
                utc_date = local_date.astimezone(pytz.utc)

                output = []
                for timezone in OTHER_TIMEZONES:
                    output.append(utc_date.astimezone(timezone))
                for appointment in output:
                    print(appointment.strftime(fmt))
                break
```

Code Challenge :
Challenge Task 1 of 1
Create a function named `to_timezone` that takes a timezone name as a string. Convert `starter` to that timezone using `pytz`'s timezones and return the new `datetime`.

```
In [ ]: import datetime

        import pytz

        starter = pytz.utc.localize(datetime.datetime(2015, 10, 21, 23, 29))

        def to_timezone(tz_str):
            return starter.astimezone(pytz.timezone(tz_str))
```