datetime

Python has the datetime module to help deal with timestamps in your code. Time values are represented with the time class. Times have attributes for hour, minute, second, and microsecond. They can also include time zone information. The arguments to initialize a time instance are optional, but the default of 0 is unlikely to be what you want.

time

Let's take a look at how we can extract time information from the datetime module. We can create a timestamp by specifying datetime.time(hour,minute,second,microsecond)

In [1]:

```
import datetime

t = datetime.time(4, 20, 1)

# Let's show the different components
print(t)
print('hour  :', t.hour)
print('minute:', t.minute)
print('second:', t.second)
print('microsecond:', t.microsecond)
print('tzinfo:', t.tzinfo)
```

```
04:20:01
hour  : 4
minute: 20
second: 1
microsecond: 0
tzinfo: None
```

Note: A time instance only holds values of time, and not a date associated with the time.

We can also check the min and max values a time of day can have in the module:

In [2]:

```
print('Earliest  :', datetime.time.min)
print('Latest    :', datetime.time.max)
print('Resolution:', datetime.time.resolution)
```

```
Earliest  : 00:00:00
Latest    : 23:59:59.999999
Resolution: 0:00:00.000001
```

The min and max class attributes reflect the valid range of times in a single day.

Dates

datetime (as you might suspect) also allows us to work with date timestamps. Calendar date values are represented with the date class. Instances have attributes for year, month, and day. It is easy to create a date representing today's date using the today() class method.

Let's see some examples:

In [3]:

```
today = datetime.date.today()
print(today)
print('ctime:', today.ctime())
print('tuple:', today.timetuple())
print('ordinal:', today.toordinal())
print('Year :', today.year)
print('Month:', today.month)
print('Day  :', today.day)
```

```
2018-02-05
ctime: Mon Feb  5 00:00:00 2018
tuple: time.struct_time(tm_year=2018, tm_mon=2, tm_mday=5, tm_hour=
0, tm_min=0, tm_sec=0, tm_wday=0, tm_yday=36, tm_isdst=-1)
ordinal: 736730
Year : 2018
Month: 2
Day  : 5
```

As with time, the range of date values supported can be determined using the min and max attributes.

In [4]:

```
print('Earliest  :', datetime.date.min)
print('Latest    :', datetime.date.max)
print('Resolution:', datetime.date.resolution)
```

```
Earliest  : 0001-01-01
Latest    : 9999-12-31
Resolution: 1 day, 0:00:00
```

Another way to create new date instances uses the replace() method of an existing date. For example, you can change the year, leaving the day and month alone.

In [5]:

```
d1 = datetime.date(2015, 3, 11)
print('d1:', d1)

d2 = d1.replace(year=1990)
print('d2:', d2)
```

```
d1: 2015-03-11
d2: 1990-03-11
```

Arithmetic

We can perform arithmetic on date objects to check for time differences. For example:

In [6]:

```
d1
```

Out[6]:

```
datetime.date(2015, 3, 11)
```

In [7]:

```
d2
```

Out[7]:

```
datetime.date(1990, 3, 11)
```

In [8]:

```
d1-d2
```

Out[8]:

```
datetime.timedelta(9131)
```

This gives us the difference in days between the two dates. You can use the timedelta method to specify various units of times (days, minutes, hours, etc.)

Great! You should now have a basic understanding of how to use datetime with Python to work with timestamps in your code!