**11장. 시간대**

같은 시간에서의 시간 계산은 재미있고 흥미롭고 지루하다. 재미 있는 것은 처음에 시간을 읽는 법을 배울 때, 여러분이 서머 타임(daylight saving)에 대해 알게될 때, 고객이 웹 애플리케이션을 통해 시간대별로 회의 일정을 잡을 수 없다고 불평할 때이다.

11장은 다음을 살펴볼 것이다.

-시간, 날짜 및 달력 작업

-날짜 결합 및 날짜 산술 비교

-날짜 형식 및 파싱

-표준 시간대 계산 처리

-표준 시간대에 따른 자동 인보이스

**소개**

"여러분이 인생을 사랑한다면, 시간을 낭비하지 말자, 시간이 인생을 만들어내는 것이기 때문이다."-브루스 리

시간은 모든 것의 척도이다. 인생에서 할 일이 너무 많으나 아이러니하게도 우리에게는 시간이 너무 부족하다. 요즘, 우리는 직관적으로 시간을 계획한다. 교통체증을 피하기 위해 몇 시에 여행해야 하는가, 다른 많은 것들 중에서 이 작업의 마감 기한은 무엇인지 등이 있다. 기업은 달력이 시작되기 전에도 1년 동안의 활동을 계획한다.

시간 계산은 거의 모든 곳에 있다. 호주에 있는 동료와 미팅 일정을 잡고 싶은가? 시간대를 정확하게 설정하고 여러분과 동료를 위해 좋은 시간을 계획한다. 적절한 시기에 고객을 위한 작업을 수행하는 코드를 작성하길 원하는가? 데이터베이스의 시간 객체를 관리하고 사용자의 모든 작업을 추적한다. 할리우드 영화인 National Treasure에서도 니콜라스 케이지(Nicholas Cage)는 시간대 계산에 의존하여 다음 단서를 찾아 보물에 더 가깝게 다가갔다.

근본적으로, 어디에 있든 무엇을 하든 시간 계산으로부터 도망칠 수는 없다. 11장에서는 파이썬의 date 객체와 time 객체를 다루게 될 것이다. 또한 날짜에 대해 산술 연산을 수행하고 시간대 계산을 사용하는 방법을 살펴본다. 사용자의 시간대를 기반으로 비즈니스 프로세스를 자동화하는 방법도 다룬다.

11장을 진행하는 동안, 내장 파이썬 모듈을 사용하도록 할 것이다. 11장에서는 다음과 같은 내장 모듈이 사용될 것이다.

-datetime(https://docs.python.org/2/library/datetime.html)

-calendar(https://docs.python.org/2/library/calendar.html)

또한 이 외부 모듈을 사용하여 시간대를 사용할 것이다.

-pytz(http://pytz.sourceforge.net/)

파이썬의 pytz 라이브러리는 Olson 시간대 데이터베이스를 가져온다. 파이썬 2로 정확하게 교차 플랫폼 시간대를 계산할 수 있다. 또한 서머 타임(daylight savings)에 관한 계산을 수행하는 데 도움이 된다.

레시피에 들어가기 전에 파이썬 설치에 관련 모듈이 있는지 확인하고 11장에서 필요한 모듈을 설치하자. 11장의 가상 환경을 생성하고 활성화하는 것으로 시작한다.

chetans-MacBookPro:ch12 Chetan$ virtualenv date

New python executable in date/bin/python2.7

Also creating executable in date/bin/python

Installing setuptools, pip, wheel...done.

chetans-MacBookPro:ch12 Chetan$ source date/bin/activate

(date)chetans-MacBookPro:ch12 Chetan$

파이썬 pip를 사용하여 가상 환경에 pytz 모듈을 설치하자. 모듈을 설치하고 나면 첫 번째 레시피로 이동하여 time 및 date 객체 작업을 시작한다.

(date)chetans-MacBookPro:ch12 Chetan$ pip install pytz

Collecting pytz

Using cached pytz-2016.7-py2.py3-none-any.whl

Installing collected packages: pytz

Successfully installed pytz-2016.7

**시간, 날짜 및 달력 작업**

이 레시피에서는 내장된 파이썬 모듈을 사용할 것이므로 명시적으로 설치할 필요가 없다. 시작하자.

**실행방법**

1. 가장 좋아하는 편집기로 이동하여 time\_ex.py라는 파일을 생성하고 파이썬 파일에 다음 코드를 작성한다.

import datetime

time\_obj = datetime.time(13, 2, 23)

print "Time object is:", time\_obj

print 'Hour :', time\_obj.hour

print 'Minute:', time\_obj.minute

print 'Second:', time\_obj.second

print 'Microsecond:', time\_obj.microsecond

위의 파이썬 코드를 실행하면 다음과 같은 출력이 나타낼 것이다. 파이썬을 사용하여 주어진 time 객체를 어떻게 생성했는지 살펴보고 주어진 time 객체에 대한 hour, minute, second, microsecond 세부 사항을 검색한다.

<그림>

2. 파이썬의 Time 클래스에는 시간 계산에 효과적으로 사용할 수 있는 몇 가지 속성이 있다. 예를들면, 다음 코드에서 주어진 날짜를 위한 유효 시간 범위를 얻을 수 있다.

import datetime

print "Time Attributes are:"

print "Earliest time of the day :", datetime.time.min

print "Latest time of the day :", datetime.time.max

앞의 코드의 출력은 다음 스크린샷에서 살펴본다. 그 날에 사용가능한 최초 시간과 마지막 시간을 얻는 방법을 살펴본다.

<그림>

3. 좋아! 이제 date 객체를 살펴보자. 다음 파이썬 코드는 오늘 날짜를 가져온다. 다음 코드로 year, month, day 속성도 검색한다.

import datetime

today = datetime.date.today()

print 'Date object:', today

print 'Year:', today.year

print 'Mon :', today.month

print 'Day :', today.day

앞의 코드의 출력은 다음 스크린샷에 표시된다.

<그림>

4. 좋아! 파이썬의 date() 메소드 및 replace() 메소드를 사용하여 새로운 date 객체를 생성하거나 기존 객체를 수정할 수도 있다. 다음 코드는 이러한 메소드를 사용하는 방법을 보여준다.

import datetime

date\_1 = datetime.date(2011, 12, 31)

print ' Date is:', date\_1

date\_2 = date\_1.replace(year=2012, month=1)

print ' New Date is:', date\_2

앞의 출력은 다음과 같다.

<그림>

5. 환상적이다! 앞으로 몇 개월 혹은 몇 년에 걸쳐 작업할 수 있는지 살펴보자. 파이썬으로 전체 일정에 대해 작업하는 것은 매우 쉽다. 다음 코드는 콘솔에서 전체 연도의 달력을 인쇄한다. 이 경우, 2017 년도의 달력을 반환한다. 내 생일을 확인했다. 오, 올해 생일은 화요일이고 나는 사무실에 가야한다.

import calendar

from calendar import TextCalendar

cal = TextCalendar()

cal.pryear(2017)

cal.prmonth(2017, 11)

앞의 코드의 출력은 다음 스크린샷에 표시된다. 첫 번째 스크린샷은 2017 년의 전체 달력을 반환한다.

<그림>

다음 스크린샷은 2017년 11월, 즉, 2017년의 11번째 월의 달력만 반환한다.

<그림>

**작동원리**

이 레시피에서 time 객체 작업을 시작했다. 시간, 분, 초를 입력 매개변수로 사용하는 datetime.time() 메소드로 time 객체를 생성했다. hour, minute 및 second 속성으로 time 객체를 읽고 datetime.time.min 및 datetime.time.max 속성으로 가장 이른 시간과 가장 늦은 시간을 얻었다.

다음으로, datetime.date() 메소드를 사용하여 날짜 작업을 시작했다. datetime.date() 메소드를 사용하여 오늘 날짜를 얻었고, today.year, today.month 및 today.day 속성을 사용하여 오늘의 년, 월, 일을 인쇄했다.

year, month 및 day를 매개변수로 전달하여 datetime.date() 메소드를 사용하여 새 날짜도 생성했다. date 객체가 사용가능해지면, replace() 메소드를 사용하여 새 날짜를 가져오는 데 사용했다. year와 month을 매개변수로 replace() 메소드에 전달하여 새 date 객체를 생성했다.

이 절에서는 calendar 객체도 사용했다. 이러한 목적으로 파이썬 설치에서 사용할 수 있는 calendar 모듈을 사용했다. 먼저 텍스트 calendar 객체를 생성하기 위해 TextCalendar 클래스를 인스턴스화한다. 나중에, 이 객체를 사용했고, pryear() 메소드로 2017년 달력을 콘솔에 인쇄했다.

또한 prmonth() 메소드를 사용하여 2017년 11월의 달력을 보여줄 수도 있다. 좋다.

**date 객체 및 time 객체 비교 및 결합, 날짜 계산**

date 객체를 생성하고 사용하는 것도 좋지만 비즈니스 사용 사례에 대해 수행해야 하는 작업은 date 및 time 객체의 차이를 비교하거나 계산하는 작업과 관련이 많다. 이 레시피에서는 파이썬에서 이러한 작업을 수행하는 방법을 다룰 것이다. 그러나 이 레시피에서 살펴볼 수 있는 중요한 변화에 주목하는 것이 중요하다. 마지막 레시피에서는 time 객체와 date 객체를 독립적으로 작업했다. 하지만 파이썬의 datetime 모듈은 날짜와 시간 속성을 모두 포함하는 객체를 작업한다는 점에서 큰 이점을 제공한다. '실행방법' 절에서 이 차이를 확인할 수 있다.

**준비하기**

이 레시피에서는 지난 몇 가지 예제에서 사용한 datetime 파이썬 모듈을 사용할 것이다. 이 레시피를 위해 새로 설치할 것은 없다.

**실행밥법**

1. 우선 두 datetime 객체의 차이를 가져온다. 다음 코드는 이 작업을 수행하고 두 datetime 객체 간의 차이를 계산한다. 이 작업은 초 단위로 차이를 알려주지만 달 혹은 년에서 차이를 구하기 위해 이 것을 사용할 수도 있다. 다음 스크린샷에서 datetime.now()가 오늘 날짜와 현재 시간이 모두 포함된 문자열을 반환하는 방법을 살펴본다. 날짜와 시간 속성을 구성하는 객체에 대해 작업한다는 것을 이해하는 것이 중요하다. 생각해 보면, 실제 세계에서도, 두 사건 간의 시간 차이를 계산해야 할 때, date 객체 및 time 객체로 함께 작업하는 것이 가장 유용할 것이다. date 객체 혹은 time 객체에 대해 독립적으로 작업하더라도, datetime 객체에 대해 수행하는 것과 동일한 계산을 수행하게 되므로 이 접근 방식으로 얻을 수 있는 이점을 상상해본다.

from datetime import datetime

import time

now\_1 = datetime.now()

print " Time Now", now\_1

time.sleep(5)

now\_2 = datetime.now()

print " Time Now", now\_2

print " Difference in the times is:", (now\_2 - now\_1).seconds

앞의 코드의 출력은 다음 스크린샷에 표시된다. 몇 초 안에 datetime 객체의 차이를 확인한다.

<그림>

2. 좋다 하지만 datetime 객체의 차이가 음수이면 어떤 일이 발생하는지 물을 수 있다. 이 예제에서 now\_1 - now\_2을 계산하면, 큰 차이가 발생하지만 실제 차이는 발생하지 않는다. 이를 위해 두 datetime 객체 간의 차이를 알아내는 멋진 요령을 안다. (now\_1 - now\_2).total\_seconds()를 사용하여 음수값, 즉 -5초를 얻을 수 있다.

3. 자, 이제 datetime 객체에 대해 더 많은 계산을 수행하자. 예를 들어, 과거에서 혹은 미래로 시간을 얻는 것은 어떨까? 다음 코드는 이러한 작업을 수행하는 데 도움이 된다. 이 순간에 어떻게 날짜와 시간을 얻었는지, 그리고 내일의 날짜와 시간을 반환할 수 있는지 살펴본다.

from datetime import datetime, timedelta

now = datetime.now()

print " Time Now is:", now

one\_day\_later = now + timedelta(days=1)

print " Tomorrow is:", one\_day\_later

앞의 코드의 출력은 다음과 같다.

<그림>

4. 과거의 시간을 얻으려면, 다음 코드와 같은 방법으로 처리될 수 있다.

from datetime import datetime, timedelta

now = datetime.now()

print " Time Now is:", now

days\_in\_past = now - timedelta(days=365, hours=1)

print " Last year:", days\_in\_past

앞의 코드의 출력은 다음 스크린샷에 표시된다. 365일 전의 과거 날짜를 요청했다. 하지만, 2015년 11월 28일을 보여준다. 왜? 같은 날을 보여 주면 안 될까? 아, 물론, 2016년은 윤년(leap year)이었다.

<그림>

5. 좋아, 이제는 차이를 얻거나 date 객체 및 time 객체에 시간을 추가하는 데 익숙해졌다. 하지만 종종 시간을 비교할 필요가 있다, 그렇지? 코드의 도움으로 이것을 살펴보자. 다음 파이썬 코드는 time 객체 및 date 객체를 개별적으로 비교한다.

import datetime

time\_1 = datetime.time(8, 9, 10)

print " Time 1:", time\_1

time\_2 = datetime.time(13, 19, 50)

print " Time 2:", time\_2

print " Comparing times: time\_2 > time\_1?", time\_2 > time\_1

date\_1 = datetime.date.today()

print " Date 1:", date\_1

date\_2 = date\_1 + datetime.timedelta(days=2)

print " Date 2:", date\_2

print " Comparing dates: date\_1 > date\_2?", date\_1 > date\_2

앞의 코드 출력은 다음 스크린샷에서 확인할 수 있다.

<그림>

6. 이전 레시피에서 보았듯이 time 객체와 date 객체를 결합해야 할 필요성도 느껴진다. 예를 들어 time 객체를 비교하고 몇 가지 작업을 수행하려는 사용 사례에 대한 프로그램을 개발했다. 하지만 비교하는 time 객체가 다른 날짜에 위치하기 때문에 날짜를 비교하게 될 수도 있다. datetime 객체를 사용하는 것이 얼마나 쉬운지 이미 알고 있기 때문에 time 객체와 date 객체를 단일 datetime 객체로 결합하여 쉽게 작업할 수 있다. 이것을 파이썬에서 쉽게 얻을 수 있다. 다음 코드는 독립 시간과 날짜를 datetime 객체와 결합하는 방법을 보여준다.

import datetime

time\_1 = datetime.time(13, 44, 55)

time\_2 = datetime.time(13, 44, 55)

print " Times:", time\_1, time\_2

date\_1 = datetime.date.today()

date\_2 = date\_1 + datetime.timedelta(days=1)

print " Dates:", date\_1, date\_2

datetime\_1 = datetime.datetime.combine(date\_1, time\_1)

datetime\_2 = datetime.datetime.combine(date\_2, time\_2)

print " Datetime Difference:", datetime\_2 - datetime\_1

앞의 코드 출력은 다음과 같다.

<그림>

**작동원리**

마지막 절에서는 time, date 및 calendar 객체를 독립적으로 작업했다. 이 레시피에서는 완전한 datetime 객체 작업을 시작했다.

이 레시피의 첫 번째 코드 예제에서 datetime 객체 간의 차이를 계산했다. 익숙했던 동일한 뺄셈(-) 연산자를 사용하면 쉽게할 수 있다. 즉, \_\_sub \_\_() 메소드가 datetime 클래스에 대해 오버라이드 됐다.

그런 다음, 두 번째 및 세 번째 코드 조각에서 timedelta() 메소드를 사용하여 미래의 datetime 객체를 얻거나 과거로 이동했다. timedelta() 메소드는 현재 datetime 객체를 과거 혹은 미래로 이동하기 위해 일 또는 시간과 같이 편리하게 명명된 속성을 지원한다. 과거에는 - timedelta()를 사용하고 + timedelta() 작업으로 이동한다.

다음으로, datetime 객체를 비교하는 방법을 찾았다. 다른 파이썬 객체와 마찬가지로 간단히 수행됐다. 파이썬에서는 정수가 각각 < and > 연산자를 가진 다른 정수보다 작거나 큰지 확인한다. datetime 객체의 경우도 마찬가지다. 이 연산자를 사용하여 datetime 객체를 비교하기만 하면 된다.

마지막으로, 차이점을 얻거나 비교하기 위해 date와 time 객체를 작업해야했던 유스 케이스를 살펴봤다. 이를 위해 파이썬 코드를 작성하여 date 및 time 객체를 결합하고 datetime.combine() 메소드를 사용했다. 이렇게하면 date 혹은 time 객체에서 개별적으로 수행한 다음 결과를 병합하는 대신 비교 또는 차이 작업을 datetime 객체에서 쉽게 수행할 수 있다.

**날짜 형식 및 파싱**

지금까지의 모든 레시피에서는 date 혹은 time 객체에 대해 여러 작업을 수행했다. 그러나 객체 자체는 특정 형식으로 표현된다. 예를 들어, 기본적으로 date() 객체는 YYYY-MM-DD 형식으로 표현되고 time() 객체는 HH:MM:SS 형식으로 표현된다. 이러한 표현이 좋지만 웹사이트의 사용자에게 데이터를 표시하거나 웹 페이지에서 회의를 예약할 때 항상 이러한 형식을 사용할 수는 없다.

이 절에서는 date와 time 객체가 사용자에게 표시될 수 있는 다양한 형식을 빠르게 살펴본다.

**준비하기**

이 레시피에서는 기본 파이썬 설치에서 패키지화된 동일한 datetime 모듈을 사용한다.

**실행방법**

1. 알고 있는 것을 시작해보자. 다음 파이썬 코드는 날짜와 시간을 ISO 형식으로 인쇄한다. 이 형식은 전세계에서 가장 많이 사용되는 형식이며 일반적으로 허용된다.

import datetime

today = datetime.datetime.today()

print " ISO datetime: ", today

2. 그러나 이미 상상했던 것처럼 이 형식은 읽기 쉽지 않다. 예를 들어, 그것은 숫자(11월 11)에서 월을 읽고 마이크로초(심지어는 별로 유용하지 않다고 생각한다)까지 시간을 반환한다. 이러한 문제를 해결하는 형식은 무엇이며 날짜를 읽기 쉽도록 만드는 방법은 어떤가? 예, 다음 코드 조각을 사용하면 쉽게할 수 있다. 이 코드에서는 '%b'와 같은 특정 형식 지정자를 사용하여 월을 읽을 수 있도록 관리한다.

import datetime

today = datetime.datetime.today()

print " ISO datetime: ", today

format = "%a %b %d %H:%M:%S %Y"

string\_format = today.strftime(format)

print " Datetime in String format:", string\_format

3. Unix timestamp 혹은 epoch를 사용하여 시간을 저장하는 웹 애플리케이션을 살펴봐야 한다. 객체를 저장하는 좋은 방법이지만 실제 시간이나 날짜를 이해하는 형식으로 사용자에게 나타내야한다.

4. 유닉스 시간은 POSIX 시간 혹은 에포크 시간이라고도하며 1970년 1월 1일 목요일 00:00:00 UTC 이후로 경과된 초로 정의된 시간을 설명하기 위한 시스템이다. Unix 타임 스탬프는 시간대와 관계없이 시간을 나타내기 때문에 유용하다. 예를 들어, 유닉스 시간은 런던에서 오후 1시, 뉴욕에서 오전 8시를 나타낼 수 있다.

다음 코드 조각은 타입스탬프를 datetime 객체와 반대로 변환하는 방법을 보여준다.

import datetime

import time

time\_1 = time.time()

print " Datetime from unix timestamp:",

datetime.datetime.fromtimestamp(1284101485)

date\_1 = datetime.datetime(2012,4,1,0,0)

print " Unix timestamp", date\_1.strftime('%s')

5. datetime 객체의 또 다른 재미있는 표현은 세계가 시작된 후 n번째 날의 날짜를 표시하는 것이다. 예를 들어, 0001년 1월 1일 이후의 1000일째 날짜를 인쇄할 수 있을까? 01/01/01에 서수 1이 있는 프로플레티시 그레고리(propleticy Gregorian)력에 해당하는 날짜이다.

import datetime

date\_1 = datetime.date.fromordinal(1000)

print " 1000th day from 1 Jan 0001: ", date\_1

위의 파이썬 코드 조각을 실행하면 다음 스크린샷과 같이 원하는 객체를 볼 수 있다.

<그림>

**작동원리**

이 레시피에서는 datetime 객체를 나타내는 다양한 방법을 살펴본다. 첫 번째 예제에서는 ISO 형식으로 날짜와 시간을 인쇄했다. 이것은 가장 많이 사용되는 형식이며 https://en.wikipedia.org/wiki/ISO\_8601에서 ISO 형식에 대한 자세한 내용을 확인할 수 있다. 보시다시피, 이 표현을 위해 새로운 메소드를 사용할 필요가 없다. 단순히 ISO 형식으로 날짜를 가져오기 위해 datetime.today()를 사용한다.

두 번째 예제에서는 문자열 형식으로 날짜를 나타내는 자체 형식을 정의하는 방법을 살펴본다. %a, %b 및 %d와 같은 형식 지정자를 사용하여 날짜와 %H, %M 및 %S로 작업하여 시간에 대한 작업을 수행했다. format 변수에 형식으로 지정하고 ISO datetime 객체를 사용자 정의 문자열 형식으로 포맷한 strftime() 메소드에 전달하는 데 사용했다.

다음 두 예제는 Unix 타임 스탬프 혹은 epoch을 datetime 객체와 vice versa로 변환하는 데 도움을 준다. 첫 번째 유스 케이스에서는 datetime.fromtimestamp(<unixtimestamp>) 메소드를 사용하여 Unix 타임 스탬프를 datetime 객체로 변환하고, 연속 예제에서 strftime() 메소드를 사용하여 datetime 객체를 Unix 타임 스탬프로 변환했다 . 이 예제에서 사용된 Unix 시간(1284101485)은 1970년 1월 1일 이후로 경과된 초 수다.

마지막으로 그리고 흥미로운 예제에서 날짜와 시간을 fromordinal() 메소드를 사용하여 그레고리안의 달력 형식으로 얻는다. 이 메소드를 사용하지는 않겠지만 11장에서 알 수 있는 흥미로운 날짜 형식으로 11장을 포함시켰다.

**표준 시간대 계산 처리**

date 혹은 time 객체에서 수행해야 하는 가장 까다로운 계산 중 하나는 시간대가 포함된 계산이다. 동료가 샌프란시스코에서 일하고 시드니에 있다면 전화 회의를 어떻게 계획하는가? 회의를 설정할 때 동료의 시간대를 알고 있어야한다. 그렇지 않으면 오후 8시에 시드니에서 회의를 설정할 수 있지만 샌프란시스코의 동료는 이미 자정이 지난 것이다. 표준 시간대 계산은 종종 지루하고 비즈니스 애플리케이션을 개발하는 동안 조심스럽게 처리해야 한다. 파이썬이 어떻게 우리를 도울 수 있는지 살펴보자.

**준비하기**

이 레시피는 11장의 시작 부분에서 설치한 pytz 모듈을 사용한다. 실제로 파이썬 표준 라이브러리에는 표준 시간대 라이브러리가 없지만 파이썬 커뮤니티에서 필요로 하는 모듈에 완전히 의존할 수 있다.

**실행방법**

다음 단계를 수행한다.

1. UTC로 현지 시간을 가져오는 간단한 작업부터 시작하자. UTC는 클럭 및 시간 측정을 규제하는 세계표준인 Universal Time Converter의 약자이다. UTC는 그리니치 표준시(GMT)로도 널리 알려졌다.

from datetime import datetime, timedelta

now = datetime.now()

print " Local time now is:", now

utcnow = datetime.utcnow()

print " UTC time now is:", utcnow

앞의 코드 조각의 출력은 다음 스크린샷에 표시된다. 내 현지 시간이 UTC보다 +5:30 시간 앞선 때를 살펴본다.

<그림>

2. 좋아, 이거 좋다. 따라서 현지 시간을 UTC로 변환할 수는 있지만 이것이 항상 충분하지는 않다. 고객(애플리케이션을 개발하는 고객)은 전 세계 어느 곳에서나 볼 수 있다. 고객의 계정은 시간대 및 현지 시간과 관련하여 관리해야 한다. 주어진 시간대에 있는 사용자의 현지 시간을 파악하는 방법을 살펴보자.

from pytz import timezone

import pytz

utc = pytz.utc

print " Selected time zone:", utc

eastern = timezone('US/Eastern')

print " Switched to time zone:", eastern

loc\_dt = datetime(2016, 11, 27, 12, 0, 0, tzinfo=pytz.utc)

est = loc\_dt.astimezone(eastern)

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

print " Local time in Eastern time zone:", est.strftime(fmt)

앞의 코드 출력은 다음 스크린샷에 표시된다. 동부 시간대를 가져와서 현지 UTC 시간을 동부 표준시(EST, Eastern Standard Time)로 변환한 방법을 살펴본다. 실제로 UTC는 시간대별로 시간을 변환하는 가장 좋은 방법이다.

<그림>

3. 파이썬에서는 표준 시간대 정보가 있는 datetime 객체로 산술 계산을 수행하는 것도 간단하다. 다음 코드를 살펴보고 date 객체에서 산술 연산을 수행하는 방법을 다룬다.

from datetime import datetime, timedelta

au\_tz = timezone('Australia/Sydney')

local = datetime(2002, 10, 27, 6, 0, 0, tzinfo=au\_tz)

print " Local time in Sydney:", local

past = local - timedelta(minutes=10)

print " 10 minutes before time was:", past

future = local + timedelta(hours=18)

print " 18 hours later it is:", future

이제 파이썬 인터프리터에서 이 코드를 실행하면 다음과 같은 결과가 나타난다.

<그림>

4. 이 문제에 대해 실제로 이야기 없이 시간대에 관한 주제를 끝낼 수 없는가? 그래, 시간대 계산에서 일광 절약 시간제는 어떻게 처리하나? 일광 절약의 세계에서 벤자민 프랭클린에게 감사의 말을 전한다. 코드 예제의 도움으로 이를 이해하자.

eastern = timezone('US/Eastern')

dt = datetime(2016, 11, 06, 1, 30, 0)

dt1 = eastern.localize(dt, is\_dst=True)

print " Date time 1 with day light savings:", dt1.strftime(fmt)

dt2 = eastern.localize(dt, is\_dst=False)

print " Date time 2 without day light savings:", dt2.strftime(fmt)

코드 조각을 실행하면 문자열 형식으로 표시된 두 datetime 객체가 나타난다. 첫 번째 것은 일광 절약 시간을 처리하고 두 번째 것은 이를 무시한다. 2016년 11월 6 일은 일광 절약 시간이 동부 시간대에서 올해 끝나고 시계가 뒤로 이동한 것이다.

<그림>

5. 마지막으로 pytz 모듈에서 사용할 수 있는 몇 가지 헬퍼 메소드가 있다. 예를 들어 ISO 국가 코드를 기반으로 특정 국가의 표준 시간대를 얻거나 단순히 ISO 국가 코드에서 국가 이름을 가져오는 것과 같이 유용할 수 있다. 다음 예제를 살펴보자.

tz\_au = '\n '.join(pytz.country\_timezones['au'])

print " Time zones in Australia:", tz\_au

country\_gb, country\_fr = pytz.country\_names['gb'],

pytz.country\_names['fr']

print "\n Country names are:\n", " ",

country\_gb, "\n ", " ", country\_gb, "\n ", country\_fr

앞의 코드 조각의 출력은 다음 스크린샷에서 확인할 수 있다.

<그림>

**작동원리**

이 레시피에서는 날짜-시간 계산에 필수적인 시간대를 사용하여 작업하는 다양한 방법을 살펴봤다. 이 레시피의 첫 번째 코드 예제에서는 datetime.now()를 사용하여 현재 로컬 시간을 계산한 다음 datetime.utcnow()를 사용하여 UTC에서 동일한 현지 시간을 얻는다. utcnow() 메소드는 일정 처리와 같은 추가 처리를 위해 데이터베이스에 date/time 객체를 저장해야할 때 매우 편리하다.

다음으로, 다른 시간대로 전환하고 해당 시간대의 현지 시간을 검색하는 방법을 살펴본다. pytz 클래스는 UTC로 표준 시간대를 설정하는 간단한 속성인 utc가 있다. 현재 시간대를 UTC로 설정하는 데 사용했다. 나중에 우리는 pyz 모듈의 timezone() 메소드를 사용하여 timezone('US/Eastern')로 전환했다.

이전의 모든 레시피에서는 datetime() 메소드를 사용하여 datetime 객체를 생성했다. 이 레시피도 datetime 메소드를 사용했지만 tzinfo 매개변수는 datetime(YYYY, MM, DD, HH, MM, SS, tzinfo=<timezone>) 형식으로 사용했다. Thet zinfo 매개변수는 표준 시간대 정보를 시간대 정보는 시간대에 따라 계산을 수행하는 데 중요한 datetime 객체에 추가한다.

datetime 클래스는 datetime 객체를 선택한 시간대(astimezone() 메소드)로 나타내는 또 다른 편리한 메소드를 가진다. 이 메소드를 사용하여 이 코드인 loc\_dt.astimezone(eastern)을 사용하여 UTC datetime 객체를 동부 표준시로 변환했다.

마지막으로, strftime(format) 메소드로 동부 시간을 표현하기 위해 커스텀 문자열 형식을 생성했다.

datetime 객체와 마찬가지로 시간대 계산 도중 시간/일을 추가하거나 제거할 수 있다. 이 레시피의 세 번째 코드 샘플에서는 Australia/Sydney 시간대로 전환하고 이 시간대에 대한 datetime 객체를 생성했다. 이 작업으로 시드니의 현지 시간으로 돌아왔다. timedelta() 메소드를 사용하여 local - timedelta(mins=10)로 현지 시간에서 10 분을 제거하고 local + timedelta(hours=18)로 18시간을 추가했다. 이렇게하면 과거 혹은 미래의 시간에 접근할 수 있다. 시간 여행으로 생각하십시오.

네 번째 코드 조각에서는 일광 절약을 사용하는 방법을 살펴봤따. 이를 이해하기 위해 표준 시간대 정보가 없는 datetime 객체를 생성하고 dt 변수에 할당했다. eastern = timezone('US/Eastern') 코드를 사용하여 동부 표준 시간대 객체를 생성했다. 그런 다음 표준 시간대 객체의 localize() 메소드를 사용하여 dt 객체를 동부 표준시로 변환했다. 다음은 다른 매개변수인 is\_dst를 localize(is\_dst=<True / False>) 메소드에 추가하여 일광 절약 시간을 고려하거나 고려하지 않고 동부 시간대의 현지 시간을 반환한다.

2016년 11월 6일은 시계가 오전 2시에 뒤로 이동한 날이었다. 따라서 예제에서는 동부 표준시 is\_dst=True로 오전 1시 30분에 질의했을 때 동부 표준시 (UTC-0400 시간)보다 4 시간 늦은 EDT(Eastern Daylight Time)으로 시간이 반환됐다. is\_dst=False로 같은 시간 동안 질의를 하면 UTC-0500 시간인 EST로 시간을 반환한다.

마지막으로 이 레시피의 예제에서 pytz 모듈이 제공하는 유용한 헬퍼 메소드를 살펴봤다. 예를 들어, pytz.country\_timezones['au']는 Au(호주)에서 사용 가능한 모든 시간대를 반환했으며 pytz.country\_names['gb']는 ISO 국가 코드 gb에 따라 영국(UK)의 이름을 반환했다. 실제로 일부 표준 시간대 문제를 해결할 때 이러한 라이브러리의 유용성을 깨닫게될 것이다.

**사용자 시간대를 기반으로 인보이스 자동화**

제이콥은 북미 Anzee Corporation의 재무 관리자로 고객 인보이스를 담당한다. Anzee Corporation은 SaaS(Software as a Service) 플랫폼을 고객에게 제공하고 플랫폼 사용에 따라 고객에게 요금을 부과한다. Anzee의 고객은 잘못된 월별 인보이스 관련 불만을 제기했다. 고객의 말로는 "지난달의 인보이스는 다음 달 1일부터 사용 가능하다. 괜찮았지만 사용량의 일부는 설명되지 않았다. 이것은 우리 회계를 엉망으로 만든다."

현재 제이콥은 플랫폼의 데이터베이스 레코드에서 고객의 지불 및 플랫폼 사용에 대한 데이터를 가져 와서 수동 방식으로 인보이스를 생성한다. 매달 고객 수가 증가함에 따라 제이콥은 수작업 프로세스가 힘들고 시간이 많이 걸릴 것이라는 점을 깨달았다. 제이콥은 또한 고객이 불평하는 문제를 누군가가 보길 원한다. 제이콥을 도울 수 있는가?

**준비하기**

이 레시피에서는 이전 레시피에서 사용한 내장 파이썬 모듈을 모두 사용하고 PDF 인보이스를 생성하기 위한 fpdf 모듈을 설치하여 제이콥이 고객을 위한 인보이스를 자동으로 준비할 필요성을 해결할 것이다. 파이썬 pip를 사용하여 모듈을 설치한다.

(date)chetans-MacBookPro:ch12 Chetan$ pip install fpdf

You are using pip version 7.1.0, however version 9.0.1 is

available.

You should consider upgrading via the

'pip install --upgrade pip' command.

Collecting fpdf

Downloading fpdf-1.7.2.tar.gz

Building wheels for collected packages: fpdf

Running setup.py bdist\_wheel for fpdf

Stored in directory: /Users/chetan/Library/Caches/pip/wheels/

c9/22/63/16731bdbcccd4a91f5f9e9bea98b1e51855a678f2c6510ae76

Successfully built fpdf

Installing collected packages: fpdf

Successfully installed fpdf-1.7.2

**실행방법**

1. 데이터베이스 레코드를 살펴보자. Anzee Corporation은 MongoDB를 사용하여 해당 월의 고객 지불 및 요금 기록을 저장한다. 이 예제에서는 Anzee Corporation의 지불 레코드가 JSON 형식으로 저장된다고 가정한다.

2. 다른 사용자 테이블처럼 ID, name, city 및 timezone과 같은 필드가 있는 모든 사용자의 목록을 포함하는 사용자 문서를 사용한다. 또한 플랫폼 서비스를 지불한 사용자의 ID, 지불 금액 및 지불 날짜가 포함된 지불에 대해 사용자가 수행한 모든 지불 기록을 보관한다.

3. 완료된 모든 지불에 대한 타임 스탬프는 UTC 형식이다. 지불과 마찬가지로, 플랫폼을 사용한 사람의 사용자 ID, 사용료에 대해 청구된 금액 및 청구된 시간을 다시 포함하는 usage 레코드도 유지한다.

users = [{"id":12, "name":"John", "city":"New York",

"timezone":"US/Eastern"},

{"id":13, "name":"Johny", "city":"Indiana",

"timezone":"US/Central"}]

#All time stamps are in UTC

payments = [{"id":12, "amount":12.00,

"created\_at":"2016-11-29T11:46:07.141Z"},

{"id":13, "amount":22.00,

"created\_at":"2016-11-30T23:46:07.141Z"},

{"id":12, "amount":5.00,

"created\_at":"2016-12-01T01:00:00.141Z"}]

usage = [{"id":12, "charge":5.00,

"created\_at":"2016-11-29T11:46:07.141Z"}]

4. 이제 모든 데이터가 완성되었으므로 인보이스 생성을 위한 코드를 작성해보자. 주어진 달 동안 사용자에 대한 지불 및 사용법을 얻는 메소드를 작성하는 것으로 시작한다. 다음 코드 조각은 이 작업을 수행한다.

user\_ids = []

user\_names = []

for usr in users:

user\_ids.append(usr["id"])

user\_names.append(usr["name"])

def get\_payments(user\_id, month):

tz = [ x for x in users if x["id"] == user\_id]

tot\_payment = 0.00

for p in payments:

dt = datetime.strptime(p["created\_at"],

'%Y-%m-%dT%H:%M:%S.%fZ')

if p["id"] == user\_id and dt.month == month:

tot\_payment += p["amount"]

return tot\_payment

def get\_usage(user\_id, month):

tz = [ x for x in users if x["id"] == user\_id]

tot\_usage = 0.00

for u in usage:

dt = datetime.strptime(u["created\_at"],

'%Y-%m-%dT%H:%M:%S.%fZ')

if u["id"] == user\_id and dt.month == month:

tot\_usage += u["charge"]

return tot\_usage

5. 다음으로 제이콥이 자신의 플랫폼을 원했던 것처럼 자동으로 PDF 인보이스를 생성하는 코드를 작성해보자. 이를 위해 fpdf 모듈을 사용한다. 다음 코드는 인보이스를 생성한다.

def get\_invoice(user\_name, user\_id, month):

html = """

<p>Anzee Corporation</p><br>

<b><p>Account Name: """ + user\_name + """</p>

<p>Invoice for month of: """ +

str(calendar.month\_name[month]) + """</p></b>

<br><br>

<p><b>Payments and Usage:</b></p>

<table align="center" width="50%">

<thead>

<tr>

<th align="left" width="50%">Charge Type</th>

<th align="right" width="50%">Amount</th>

</tr>

</thead>

<tbody>

<tr>

<td>Payments Done</td>

<td align="right">$""" + str(get\_payments

(user\_id, month)) + """</td>

</tr>

<tr>

<td>Total Usage</td>

<td align="right">$""" + str(get\_usage

(user\_id, month)) + """</td>

</tr>

</tbody>

</table>

<br><br>

"""

return html

class MyFPDF(FPDF, HTMLMixin):

pass

html = get\_invoice("John", 12, 11)

pdf=MyFPDF()

pdf.add\_page()

pdf.write\_html(html)

pdf.output('invoice.pdf','F')

6. 앞의 코드 조각 전체를 실행하면 생성된 인보이스가 다음 스크린샷과 같다.

<그림>

7. 좋아, 이거 멋지다! 제이콥이 예상한대로 인보이스를 생성할 수 있다. 이제 완전한 프로세스를 자동화함으로써 그를 위해 많은 시간을 절약할 수 있다. 그러나 그는 또한 정확한 정보가 포함되지 않은 인보이스와 관련된 고객 불만 사항을 조사하기를 원했다. 무슨 일이 일어났는지 살펴보자.

8. 이제 Anzee Corporation은 UTC로 모든 트랜잭션과 타임 스탬프를 저장하므로 사용자가 시간을 검색하여 사용자의 시간대를 기준으로 사용자에게 보여주기가 매우 쉽다. 따라서 인보이스에 대한 해당 월의 트랜잭션을 얻기 위해 모든 레코드를 조회하면 사용자의 시간대가 아닌 UTC 타임 스탬프가 표시된다.

9. 예를 들어 JSON 데이터를 자세히 살펴보면 2016-12-01T01:00:00.141Z와 같이 created\_at 타임 스탬프가 있는 사용자 ID 12 인 John에 의한 다른 지불에 대한 언급이 있다. 이 시간은 UTC 관점에서 볼 때 11월 미만이 될 수는 없지만 지불한 사용자는 미국/동부 표준 시간대에 속한다. 따라서 UTC의 2016년 12월 1일 오전 1시는 동부 시간대의 11월 30일 오후 8시에 실제로 발생한다. 분명히 사용자는 인보이스에 자신의 지불 기능을 찾지 못한다.

10. 다음 코드 조각은 사용자 시간대를 기반으로 인보이스를 생성하여 문제를 해결한다.

from datetime import datetime

import pytz

from pytz import timezone

from fpdf import FPDF, HTMLMixin

import calendar

users = [{"id":12, "name":"John",

"city":"New York", "timezone":"US/Eastern"},

{"id":13, "name":"Johny",

"city":"Indiana", "timezone":"US/Central"}]

#All time stamps are in UTC

payments = [{"id":12, "amount":12.00,

"created\_at":"2016-11-29T11:46:07.141Z"},

{"id":13, "amount":22.00,

"created\_at":"2016-11-30T23:46:07.141Z"},

{"id":12, "amount":5.00,

"created\_at":"2016-12-01T01:00:00.141Z"}]

usage = [{"id":12, "charge":5.00,

"created\_at":"2016-11-29T11:46:07.141Z"}]

user\_ids = []

user\_names = []

for usr in users:

user\_ids.append(usr["id"])

user\_names.append(usr["name"])

def get\_payments(user\_id, month):

tz = [ x for x in users if x["id"] == user\_id]

tot\_payment = 0.00

for p in payments:

dt = datetime.strptime(p["created\_at"],

'%Y-%m-%dT%H:%M:%S.%fZ')

dt = dt.replace(tzinfo=pytz.UTC)

dt = dt.astimezone(timezone(tz[0]["timezone"]))

if p["id"] == user\_id and dt.month == month:

tot\_payment += p["amount"]

return tot\_payment

def get\_usage(user\_id, month):

tz = [ x for x in users if x["id"] == user\_id]

tot\_usage = 0.00

for u in usage:

dt = datetime.strptime(u["created\_at"],

'%Y-%m-%dT%H:%M:%S.%fZ')

dt = dt.replace(tzinfo=pytz.UTC)

dt = dt.astimezone(timezone(tz[0]["timezone"]))

if u["id"] == user\_id and dt.month == month:

tot\_usage += u["charge"]

return tot\_usage

def get\_invoice(user\_name, user\_id, month):

html = """

<p>Anzee Corporation</p><br>

<b><p>Account Name: """ + user\_name + """</p>

<p>Invoice for month of: """ +

str(calendar.month\_name[month]) + """</p></b>

<br><br>

<p><b>Payments and Usage:</b></p>

<table align="center" width="50%">

<thead>

<tr>

<th align="left" width="50%">Charge Type</th>

<th align="right" width="50%">Amount</th>

</tr>

</thead>

<tbody>

<tr>

<td>Payments Done</td>

<td align="right">$""" + str(get\_payments(

user\_id, month)) + """</td>

</tr>

<tr>

<td>Total Usage</td>

<td align="right">$""" + str(get\_usage(

user\_id, month)) + """</td>

</tr>

</tbody>

</table>

<br><br>

"""

return html

class MyFPDF(FPDF, HTMLMixin):

pass

html = get\_invoice("John", 12, 11)

pdf=MyFPDF()

pdf.add\_page()

pdf.write\_html(html)

pdf.output('invoice.pdf','F')

앞의 코드 조각에 대한 출력은 다음 스크린샷에 표시된다. 지급 컬럼에 올바른 데이터가 반영된 방법을 확인하고 2016년 11월 30일 오후 8시에 완료된 $5 지급액을 포함하여 총 금액을 $12 + $5 = $17로 계산한다.

<그림>

**작동원리**

먼저 제이콥의 인보이스 생성 자동화에 대해 살펴봤다. 모든 사용자에 대한 JSON 데이터를 파싱하고 11월 한 달 동안 모든 사용자에 대한 지불 및 사용량을 계산했다.

우리는 지불 및 사용 기록을 검토하기 위해 get\_payments(user\_id, month) 및 get\_usage(user\_id, month)를 개발하고 11월의 기록을 선택했다. created\_at JSON 문자열로 작업하고 dt = datetime.strptime(u["created\_at"], '%Y-%m-%dT%H:%M:%S.%fZ')와 함께 date/time 객체로 변환하여이 작업을 수행했다.

그러나 이전 절에서 이해했듯이 문자열을 타임 스탬프로 변환하는 것만으로는 도움이되지 않았다. 사용자의 시간대와 관련하여 시간을 고려하지 않았기 때문이다. 이를 위해 dt.replace(tzinfo = pytz.UTC)를 사용하여 date/time 객체인 dt를 UTC 시간대로 변환한 다음 dt를 사용하여 dt.astimezone(timezone(<>)) 메소드로 사용자 표준 시간대의 시간을 반영하도록 변환한다 . 이렇게하면 사용자 시간대의 지불 시간을 얻을 수 있고 인보이스 데이터는 11월의 정확한 수치를 반영한다.

다음으로, 적절한 사용자 이름, 인보이스 시간을 추가하여 인보이스에 대한 HTML 콘텐츠를 생성하고 이것을 html 변수에 저장했다. 나중에 fpdf 모듈에서 FPDF 및 HTMLMixin 클래스를 상속한 MyFPDF 클래스를 생성했다. 그런 다음 MyFPDP 클래스를 사용하여 빈 PDF 파일 객체를 나타내는 pdf 객체를 생성했다. add\_page() 메소드로 pdf 객체의 페이지를 추가하고 write\_html(html) 메소드로 HTML 컨텐츠(인보이스 컨텐츠)로 업데이트 했다. 결국 우리는 output(<filename>) 메소드를 사용하여 디스크의 모든 데이터가 포함된 pdf 객체를 덤프했다.

**더 알아보기**

파이썬에서 시간 및 시간대 작업과 관련된 다른 많은 흥미로운 사용 사례가 있다. 마지막 예제에서 보았듯이 잘 사용하지 않으면 까다로워질 수 있다. 일반적인 지침으로 추천한다.

-항상 시간대를 인식하는 datetime 객체를 사용한다. 이 접근법으로는 결코 잘못될 수 없다. 항상 알림으로 사용된다.

-지정된 객체에 대한 표준 시간대 정보를 반환하는 ISO 형식의 datetime을 반환한다.

11장을 좋아하고 예제를 즐기기를 바란다! 계속 지켜보자.