# PYTHON
# 날짜

Moon Yong Joon

# CALENDAR MODULE

# calendar

# 달력 : calendar 모듈

## 달력에 대한 정보

```
import calendar

#print(calendar.calendar(2017))
print(calendar.prmonth(2017, 1))
# 월요일은 0
print(calendar.weekday(2017, 1, 2))
# 2017년 1월의 1일은 일요일이고, 이 달은 31일까지
print(calendar.monthrange(2017,1))
```

```
    January 2017
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
 None
0
(6, 31)
```

# PYTZ MODULE

# Time zone

# Time 용어 정리

## 시간 기준에 대한 주요 용어

| class | Description |
| --- | --- |
| Time Stamp | epoch 시간(1970년 1월 1일 자정)이후로 즉 Unix 가 탄생한 사건을 기준으로 초 단위로 측정한 절대 시간 |
| UTC(Universal Time Coordinated) | 1972년부터 시행된 국제 표준시(협정 세계시)이면 세슘 원자의 진동수에 의거한 초의 길이가 기준 |
| GMT(Greenwich Mean Time) | 영국 런던의 그리니치 천문대의 자오선상을 기준으로 하는 평균 태양시 |
| LST(Local Standard Time) | UTC 기준으로 경도 15도마다 1시간 차이가 발생하는 시간이며 지방 표준시라 부른다<br>한국은 동경 135 기준으로 UTC 보다 9시간 빠르다. |
| DST(Daylight Saving Time) | 일광절약시간제로 서머타임이라고 불리며 에너지 절약을 목적으로 한 시간을 앞으로 당기거나 뒤로 미루는 제도 |

# Pytz 모듈: timezone 관리

# pytz : all_timezones

| variable | Description |
| --- | --- |
| all_timezones | pytz.lazy.LazyList으로써 모든 타임존을 가진 list |

```
import pytz

for tz in pytz.all_timezones :
    if "Asia" in tz :
        print(tz)
```

```
Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
Asia/Aqtobe
Asia/Ashgabat
Asia/Ashkhabad
Asia/Baghdad
Asia/Bahrain
Asia/Baku
Asia/Bangkok
Asia/Barnaul
Asia/Beirut
Asia/Bishkek
```

# pytz : all_timezones_set

| variable | Description |
| --- | --- |
| all_timezones_set | pytz.lazy.LazySet으로써 모든 타임존을 가진 set |

```
import pytz

for tz in pytz.all_timezones_set :
    if "Asia" in tz :
        print(tz)
```

```
Asia/Bishkek
Asia/Barnaul
Asia/Aqtobe
Asia/Srednekolymsk
Asia/Harbin
Asia/Taipei
Asia/Baku
Asia/Yakutsk
Asia/Vientiane
Asia/Gaza
Asia/Calcutta
Asia/Shanghai
Asia/Tokyo
Asia/Yerevan
Asia/Sakhalin
Asia/Dhaka
Asia/Jerusalem
Asia/Omsk
Asia/Kabul
```

# pytz : timezone에 대한 속성

| variable | Description |
| --- | --- |
| common_timezones | pytz.lazy.LazyList으로써 일반적인 타임존을 가진 리스트 |
| common_timezones_set | pytz.lazy.LazySet으로써 일반적인 타임존을 가진 set |

```
import pytz

print("Asia/Seoul" in pytz.all_timezones_set)
print("Asia/Seoul" in pytz.common_timezones)
print("Asia/Seoul" in pytz.common_timezones_set)
```

```
True
True
True
```

# pytz : country_names

국가코드로 국가 이름을 조회

```python
import pytz

print(pytz.country_timezones('KR'))
print(pytz.country_names['kr'])
print(pytz.country_timezones('US'))
print(pytz.country_names['US'])
```

```
['Asia/Seoul']
Korea (South)
['America/New_York', 'America/Detroit', 'America/Kentucky/Louisvi
lis', 'America/Indiana/Vincennes', 'America/Indiana/Winamac', 'Am
Indiana/Vevay', 'America/Chicago', 'America/Indiana/Tell_City', '
ta/Center', 'America/North_Dakota/New_Salem', 'America/North_Dako
x', 'America/Los_Angeles', 'America/Anchorage', 'America/Juneau',
rica/Nome', 'America/Adak', 'Pacific/Honolulu']
United States
```

# pytz : country_timezones

국가코드로 타임존을 조회

```
import pytz

print(pytz.country_timezones('KR'))
print(pytz.country_timezones('US'))
```

```
['Asia/Seoul']
['America/New_York', 'America/Detroit', 'America/Kentucky/Louisville
lis', 'America/Indiana/Vincennes', 'America/Indiana/Winamac', 'Ameri
Indiana/Vevay', 'America/Chicago', 'America/Indiana/Tell_City', 'Ame
ta/Center', 'America/North_Dakota/New_Salem', 'America/North_Dakota/
x', 'America/Los_Angeles', 'America/Anchorage', 'America/Juneau', 'A
rica/Nome', 'America/Adak', 'Pacific/Honolulu']
```

# 타임존 생성

# pytz : timezone

timezone으로 하나의 타임존 인스턴스를 생성하는 함 수

```
import pytz

for tzs in pytz.all_timezones :
    if 'Asia/Seoul' in tzs :
        tzs_seoul = tzs
        break

a = pytz.timezone(tzs_seoul)
print(type(a))
```

```
<class 'pytz.tzfile.Asia/Seoul'>
```

# Pytz.UTC 적용

# Pytz.UTC : 기본UTC 적용

## pytz.UTC 이해하기


```python
import pytz

print(type(pytz.UTC))
print(pytz.UTC.__class__.__class__)
print(pytz.UTC.dst(datetime.datetime.today()))
print(pytz.UTC.localize(datetime.datetime.today()))
print(pytz.UTC.tzname(datetime.datetime.today()))
print(pytz.UTC.utcoffset(datetime.datetime.today()))
print(pytz.UTC.zone)
```

```
<class 'pytz.UTC'>
<class 'type'>
0:00:00
2017-01-02 12:41:23.985054+00:00
UTC
0:00:00
UTC
```

# Pytz모듈 -UTC class 실행

UTC class는 슈퍼클래스 datetime.tzinfo의 subclass로써 실제 변수와 메소드를 구현하여 사용


```
import pytz
import datetime

print( "pytz.UTC ", type(pytz.UTC), pytz.UTC)
print( "pytz.UTC.zone ", type(pytz.UTC.zone), pytz.UTC.zone)
print( "pytz.UTC.tzname ", pytz.UTC.tzname)
print( "pytz.UTC formutc method ", pytz.UTC.fromutc(datetime.datetime.utcnow()))
```

```
pytz.UTC  <class 'pytz.UTC'> UTC
pytz.UTC.zone  <class 'str'> UTC
pytz.UTC.tzname  <bound method UTC.tzname of <UTC>>
pytz.UTC formutc method  2017-01-02 02:23:18.681715+00:00
```

# Pytz모듈 -UTC instance 실행

## UTC의 인스턴스utc에 대한 메소드 처리

| pytz.UTC |
| --- |
| @classvariable<br>zone : str<br>tzname:str |
| @classmethod<br>fromutc(dt) : str<br><br>@instancemethod<br>dst(self, dt): str<br>fromutc() : str<br>localize(self,dt) : str<br>normalize() : str<br>utcoffset() : str |

```
import pytz
import datetime

utc = pytz.utc
print( "pytz.utc dst method ", utc.dst(datetime.datetime.utcnow()))
print( "pytz.utc utcoffset method ", utc.utcoffset(datetime.datetime.utcnow()))
print( "pytz.utc localize method ", utc.localize(datetime.datetime.utcnow()))
print("pytz.utc normalize method ", utc.normalize(datetime.datetime.now(tz=pytz.timezone("Asia/Seoul"))))
```

```
pytz.utc dst method  0:00:00
pytz.utc utcoffset method  0:00:00
pytz.utc localize method  2017-01-02 02:24:35.877315+00:00
pytz.utc normalize method  2017-01-02 02:24:35.877315+00:00
```

# 타임존을 datetime에 적용

# Datetime에 Timezone 적용

## Pytz 모듈을 이용하여 타임존 가져오기

```
import pytz
import datetime

#타임존을 가져오기
for tzs in pytz.all_timezones :
    if 'Asia/Seoul' in tzs :
        tzs_seoul = tzs
    if 'America/New_York' in tzs :
        tzs_newyork = tzs

#뉴욕 타임 가져오기
print(' tzs new york ' )
EST = pytz.timezone(tzs_newyork)
print(" ETS  type ", type(EST))
dt1 = datetime.datetime.now(tz=EST)
print(' tzs newyork', dt1)

#서울 타임가져오기
EST2 = pytz.timezone(tzs_seoul)
print(" ETS 2 type ", type(EST2))
dt2 = datetime.datetime.now(tz=EST2)
print(" seoul date time ", dt2)
```

```
tzs new york
ETS  type  <class 'pytz.tzfile.America/New_York'>
tzs newyork 2017-01-01 20:46:28.299948-05:00
ETS 2 type  <class 'pytz.tzfile.Asia/Seoul'>
seoul date time  2017-01-02 10:46:28.300948+09:00
```

1. Pytz 모듈 내의 all_timezones에 세계 타임존이 있다.
2. Timezone 을 처리하기 위해 pytz.timezone(string)을 넣고 인스턴스생성
3. datatime.datetime.now(tz=pytz.timezone())을 넣고 타임존 시간을 가져옴

# 타임존 적용 및 offset 확인

한국이 정확히는 +8:30분 위치에 있기는 하지만 일본과 같이 +9:00를 사용 중이다. 그런데 위와 같은 코드를 작성하면 +8:30(테스트상 8;28차이)

```
import pytz
from datetime import datetime

tz = pytz.timezone('Asia/Seoul')
print(datetime(2017, 1, 2, tzinfo=tz))

print(datetime(2017, 1, 2, tzinfo=tz).utcoffset())

tz1 = pytz.timezone('Asia/Tokyo')
print(datetime(2017, 1, 2, tzinfo=tz1))

print(datetime(2017, 1, 2, tzinfo=tz1).utcoffset())
```

```
2017-01-02 00:00:00+08:28
8:28:00
2017-01-02 00:00:00+09:00
9:00:00
```

# pytz :offset간의 차

타임존별간의 차를  확인

```
import pytz
from datetime import datetime

tz = pytz.timezone('Asia/Seoul')
d1 = datetime(2013, 2, 3, tzinfo=tz)

print(d1)
print(d1.utcoffset())

tz1 = pytz.timezone('US/Eastern')
d2 = datetime(2013, 2, 3, tzinfo=tz1)

print(d2)
print(d2.utcoffset())

print(d1-d2)
```

```
2013-02-03 00:00:00+08:28
8:28:00
2013-02-03 00:00:00-04:56
-1 day, 19:04:00
-1 day, 10:36:00
```

# Timezone:localize 적용

# timezone : localize

timezone 내의 localize 메소드를 이용해서 datetime 생성

```
from pytz import timezone
from datetime import datetime

korea = timezone('Asia/Seoul')
print(type(korea))
print(korea.tzname)
print(korea.utcoffset)
print(korea.zone)
loc_dt = korea.localize(datetime(2017,1,2))
print(loc_dt)
print(type(loc_dt))
print(korea.normalize(loc_dt))
```

```
<class 'pytz.tzfile.Asia/Seoul'>
<bound method DstTzInfo.tzname of <DstTzInfo 'Asia/Seoul' LMT+8:28:00 STD>>
<bound method DstTzInfo.utcoffset of <DstTzInfo 'Asia/Seoul' LMT+8:28:00 STD>>
Asia/Seoul
2017-01-02 00:00:00+09:00
<class 'datetime.datetime'>
2017-01-02 00:00:00+09:00
```

# Pytz모듈 -지역시간 산출

localize() 메소드는 시간대 보정이 없는, 순수한 datetime을 지역화하는데 사용함

| pytz.tzfile.Asia/Seoul |
| --- |
| @classvariable<br>zone : str<br>tzname:str |
| @classmethod<br>fromutc(dt) : str<br><br>@instancemethod<br>dst(self, dt): str<br>fromutc() : str<br>localize(self,dt) : str<br>normalize() : str<br>utcoffset() : str |

```
import pytz
import datetime

korean = pytz.timezone('Asia/Seoul')

print( type(korean))

print( " Asia/Seoul time zone ", korean.localize(datetime.datetime.now()))
```

```
<class 'pytz.tzfile.Asia/Seoul'>
 Asia/Seoul time zone  2017-01-02 11:25:59.278215+09:00
```

실제는 Asia/Tokyo 시간 사용

# Timezone 이용 datatime 생성

localize 함수를 이용해서 datetime 생성 및 datetime.astimezone(pytz.timezone) 를 매칭하여 타임존을 변경

```
import pytz
import datetime

korean = pytz.timezone('Asia/Seoul')

print(type(korean))

print(" Asia/Seoul time zone ", korean.localize(datetime.datetime.now()))
korean_dt = korean.localize(datetime.datetime.now())
print("datatime.datetime.astimezone normalize ")
print("type ", korean_dt.astimezone(pytz.timezone('US/Eastern')))
eastern = korean_dt.astimezone(pytz.timezone('US/Eastern'))
print(" time zone change ", eastern)
```

```
<class 'pytz.tzfile.Asia/Seoul'>
 Asia/Seoul time zone  2017-01-02 11:21:45.494015+09:00
datatime.datetime.astimezone normalize
type  2017-01-01 21:21:45.494015-05:00
 time zone change  2017-01-01 21:21:45.494015-05:00
```

# Timezone:normalize 적용

# timezone : normalize

타 timezone 적용을 위해  normalize 메소드를 이용해서 datetime 변경

```
from pytz import timezone
from datetime import datetime

th = timezone('Asia/Bangkok')
am = timezone('Europe/Amsterdam')
dt = th.localize(datetime(2011, 5, 7, 1, 2, 3))
print(dt)
fmt = '%Y-%m-%d %H:%M:%S %Z (%z)'
print(am.normalize(dt).strftime(fmt))
```

```
2011-05-07 01:02:03+07:00
2011-05-06 20:02:03 CEST (+0200)
```

# localize → normalize 순서 준수

tzinfo 즉 timezone이 세팅된 후에 처리되므로 반드시 localize한 후에 처리할 것

```
import pytz
import datetime

# 타임존을 만들고
a = pytz.timezone('Asia/Seoul')
# datetime을 넣고 타임존에 맞춰 datetime을 생성
print(datetime.datetime.now().tzinfo)
asloct = a.localize(datetime.datetime.now())
print(type(asloct), asloct.tzinfo)

#utc 시간으로 생성
asloct1 = a.localize(datetime.datetime.utcnow())
print(asloct1)

b = pytz.timezone('Asia/Tokyo')
#normalize는 tzinfo가 setting 된 경우만 처리됨
print(b.normalize(asloct))
print(b.normalize(b.localize(datetime.datetime.now())))
```

```
None
<class 'datetime.datetime'> Asia/Seoul
2017-01-02 07:17:55.764219+09:00
2017-01-02 16:17:55.764219+09:00
2017-01-02 16:17:55.764219+09:00
```

# 타임존별 시간 적용

# 평양시간 : timezone 미적용

Localize 함수로 현재 타임존이 적용되므로 타 타임존(평양)이 미적용

```
import pytz
# local time 구하기
# 서울 date time 구하기
seoul_tz = pytz.timezone('Asia/Seoul')
print( " seoul time zone ",seoul_tz)
seoul_dt = seoul_tz.localize(datetime.datetime.now())
print( " seoul datetime ", seoul_dt)

# 평양 date time으로  구하기
pyongyang_tz = pytz.timezone('Asia/Pyongyang')
print( " pyongyang time zone ", pyongyang_tz)
pyongyang_dt = pyongyang_tz.localize(datetime.datetime.now())
print( " pyongyang datetime ", pyongyang_dt)
```

```
 seoul time zone  Asia/Seoul
 seoul datetime  2017-01-02 15:09:42.038008+09:00
 pyongyang time zone  Asia/Pyongyang
 pyongyang datetime  2017-01-02 15:09:42.039008+08:30
```

# 평양 시간 조정하기: timedelta

타임존 적용하지 않고 timedelta로 시간을 빼기(30분)

```
import pytz

# 평양 date time으로  구하기
pyongyang_tz = pytz.timezone('Asia/Pyongyang')
pyongyang_dt = pyongyang_tz.localize(datetime.datetime.now())
print( " pyongyang datetime ", pyongyang_dt)
before = pyongyang_tz.normalize(pyongyang_dt - datetime.timedelta(minutes=30))
print( " 실제 평양 시간 조정 ", before)
```

```
pyongyang datetime  2017-01-02 15:20:31.744508+08:30
실제 평양 시간 조정  2017-01-02 14:50:31.744508+08:30
```

# 평양 시간 조정하기 : pytz.utc

타임존이 Asia/Pyongyang을 적용하기 위해 normalize 함수를 사용해서 30분 조정

```
import pytz

# 평양 date time으로  구하기
pyongyang_tz = pytz.timezone('Asia/Pyongyang')
pyongyang_dt = pyongyang_tz.localize(datetime.datetime.now())

print( " pyongyang datetime ", pyongyang_dt)
# 평양 date time를 UTC로 구하기
print("기준시간 : ", datetime.datetime.now(tz=pytz.utc))
after = pyongyang_tz.normalize(datetime.datetime.now(tz=pytz.utc))
print( " 실제 평양시간 조정", after)
```

```
 pyongyang datetime  2017-01-02 15:19:39.629908+08:30
기준시간 :  2017-01-02 06:19:39.629908+00:00
 실제 평양시간 조정 2017-01-02 14:49:39.630908+08:30
```

# TIME MODULE

# time.struct_time class

# time.struct_time

## gmtime(), localtime(), strptime()함수의 처리 결과

```
import time
loc = time.localtime()
print(loc.tm_year)
print(loc.tm_mon)
print(loc.tm_mday)
print(loc.tm_wday)
print(loc.tm_yday)
print(loc.tm_hour)
print(loc.tm_min)
print(loc.tm_sec)

print(loc.tm_isdst)
```

```
2017
1
2
0
2
10
3
34
0
```

| Index | Attribute | Values |
|---|---|---|
| 0 | tm_year | (for example, 1993) |
| 1 | tm_mon | range [1, 12] |
| 2 | tm_mday | range [1, 31] |
| 3 | tm_hour | range [0, 23] |
| 4 | tm_min | range [0, 59] |
| 5 | tm_sec | range [0, 61] |
| 6 | tm_wday | range [0, 6], Monday is 0 |
| 7 | tm_yday | range [1, 366] |
| 8 | tm_isdst | 0, 1 or -1; see below |

# 주요 함수 처리

# 주요 함수 처리 기준

## 주요 함수 처리 기준

| From | To | Use |
| --- | --- | --- |
| seconds since the epoch | `struct_time` in UTC | `gmtime()` |
| seconds since the epoch | `struct_time` in local time | `localtime()` |
| `struct_time` in UTC | seconds since the epoch | `calendar.timegm()` |
| `struct_time` in local time | seconds since the epoch | `mktime()` |

# Time Stamp

## 시간 기준에 대한 주요 용어

| class | Description |
| --- | --- |
| Time Stamp | epoch 시간(1970년 1월 1일 자정)이후로 즉 Unix 가 탄생한 사건을 기준으로 초 단위로 측정한 절대 시간 |

```
import time
print(time.time())
```

```
1483335391.0532677
```

```
help(time.time)
```

```
Help on built-in function time in module time:

time(...)
    time() -> floating point number

    Return the current time in seconds since the Epoch.
    Fractions of a second may be present if the system clock provides them.
```

# Function : time

## 주요 함수

| method | Description |
|---|---|
| time.time() | epoch 를 기준으로 초를 float 단위로 반환 |
| time.asctime([t]) | gmtime()/localtime() –struct_time 객체를 인자로 받아 '요일문자축약 달문자축약 일자 시간:분:초 년도' 문자열로 반환 |
| time.ctime([sec]) | epoch 를 기준으로 초를 입력을 받아 '요일문자축약 달문자축약 일자 시간:분:초 년도' 문자열로 반환 |

```
import time

t= time.time()
print(time.ctime(t))

l= time.localtime()
print(time.asctime(l))
```

```
Mon Jan  2 09:28:42 2017
Mon Jan  2 09:28:42 2017
```

# UTC(Universal TimeCoordinated)

## 시간 기준에 대한 주요 용어

| class | Description |
| --- | --- |
| UTC(Universal Time Coordinated) | 1972년부터 시행된 국제 표준시(협정 세계시)이면 세슘 원자의 진동수에 의거한 초의 길이가 기준 |

```
import time
print(time.gmtime())
```

```
time.struct_time(tm_year=2017, tm_mon=1, tm_mday=2, tm_hour=5, tm_min=38, tm_sec=28, tm_wday=0, tm_yday=2, tm_isdst=0)
```

```
help(time.gmtime)
```

```
Help on built-in function gmtime in module time:

gmtime(...)
    gmtime([seconds]) -> (tm_year, tm_mon, tm_mday, tm_hour, tm_min,
                          tm_sec, tm_wday, tm_yday, tm_isdst)

    Convert seconds since the Epoch to a time tuple expressing UTC (a.k.a.
    GMT).  When 'seconds' is not passed in, convert the current time instead.

    If the platform supports the tm_gmtoff and tm_zone, they are available as
    attributes only.
```

# Function : gmtime

## 시간 기준에 대한 주요 용어

| method | Description |
|---|---|
| time.gmtime([sec]) | 입력된 초를 UTC 기준의 struct_time 객체로 변환<br>단 인자가 없을 경우 time.time() 함수 결과를 이용해 struct_time으로 변환 |

```
import time
# 현재 기준

t = time.gmtime()
print(t)
# 특정 문자열로 전환
print(time.asctime(t))
# epoch 기준으로 전환
print(time.mktime(t))
```

```
time.struct_time(tm_year=2017, tm_mon=1,
tm_mday=2, tm_hour=0, tm_min=53, tm_sec=9,
tm_wday=0, tm_yday=2, tm_isdst=0)
 Mon Jan 2 00:53:09 2017
1483285989.0
```

# LST(Local Standard Time)

## 시간 기준에 대한 주요 용어

| class | Description |
| --- | --- |
| LST(Local Standard Time) | UTC 기준으로 경도 15도마다 1시간 차이가 발생하는 시간이며 지방 표준시라 부른다<br>한국은 동경 135 기준으로 UTC 보다 9시간 빠르다. |

```
import time

a = time.localtime()
print(a)

b = time.ctime()
print(b)
```

```
time.struct_time(tm_year=2017, tm_mon=1, tm_mday=2, tm_hour=14, tm_min=59, tm_sec=21,
Mon Jan  2 14:59:21 2017
```

# Function : localtime

## 시간 기준에 대한 주요 용어

| method | Description |
| --- | --- |
| time.localtime([secs]) | 입력된 초를 지방표준시 기준의 struct_time 객체로 변환<br>단, 인자가 없을 경우 time.time() 함수 결과를 이용해 struct_time 객체로 변환 |

```
import time
# 현재 기준
l = time.localtime()

print(l)
# 1970년 기준
l1 = time.localtime(360)
print(l1)
```

time.struct_time(tm_year=2017, tm_mon=1, tm_mday=2, tm_hour=9, tm_min=50, tm_sec=25, tm_wday=0, tm_yday=2, tm_isdst=0)

time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=9, tm_min=6, tm_sec=0, tm_wday=3, tm_yday=1, tm_isdst=0)

# Function : mktime

## 시간 기준에 대한 주요 용어

| method | Description |
| --- | --- |
| time.mktime(t) | 지방표준시 기준으로 struct_time 객체를 인자로 받아 time()과 같은 누적된 초로 반환 |

```
import time

a = time.localtime()
print(a)

b = time.mktime(a)
print(b)
```

```
time.struct_time(tm_year=2017, tm_mon=1, tm_mday=2, tm_hour=15, tm_min=2, tm_sec=38,
1483336958.0
```

# Function : sleep

## 주요 함수

| method | Description |
| --- | --- |
| time.sleep(secs) | 진행 중인 프로세스를 정해진 초만큼 정지 |

# formatting

# Function : strftime

struct_time 을 받아 특정 문자열로 전환

```
import time

now = time.localtime(time.time())

print(time.asctime(now))
print(time.strftime("%y/%m/%d %H:%M", now))
print(time.strftime("%a %b %d", now))
print(time.strftime("%c", now))
print(time.strftime("%I %p", now))
```

```
Mon Jan  2 08:58:23 2017
17/01/02 08:58
Mon Jan 02
Mon Jan  2 08:58:23 2017
08 AM
```

# Function : strptime

## 문자열로 받아 특정 format으로 전환

```
import time

print(time.strptime("31 12 2000", "%d %m %Y"))

print(time.strptime("30 Nov 2000", "%d %b %Y"))

print(time.strptime("1 Jan 70 1:30pm", "%d %b %y %I:%M%p"))
```

time.struct_time(tm_year=2000, tm_mon=12, tm_mday=31, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=6, tm_yday=366, tm_isdst=-1)
time.struct_time(tm_year=2000, tm_mon=11, tm_mday=30, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=335, tm_isdst=-1)
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=13, tm_min=30, tm_sec=0, tm_wday=3, tm_yday=1, tm_isdst=-1)

# Time format – 1

| Directive | Meaning | Example |
|---|---|---|
| %a | 요일 이름 약자 | Sun, Mon, ..., Sat (en_US);<br>So, Mo, ..., Sa (de_DE) |
| %A | 요일 전체 이름 | Sunday, Monday, ..., Saturday (en_US);<br>Sonntag, Montag, ..., Samstag (de_DE) |
| %b | 월 이름 약자 | Jan, Feb, ..., Dec (en_US);<br>Jan, Feb, ..., Dez (de_DE) |
| %B | 월 전체 이름 | January, February, ..., December (en_US);<br>Januar, Februar, ..., Dezember (de_DE) |
| %c | Locale's appropriate date and time representation. | Tue Aug 16 21:30:00 1988 (en_US);<br>Di 16 Aug 21:30:00 1988 (de_DE) |
| %d | day as a zero-padded decimal number. | 01, 02, ..., 31 |
| %H | Hour (24-hour clock) as a zero-padded decimal number. | 00, 01, ..., 23 |
| %I | Hour (12-hour clock) as a zero-padded decimal number. | 01, 02, ..., 12 |
| %j | Day of the year as a zero-padded decimal number. | 001, 002, ..., 366 |
| %m | Month as a zero-padded decimal number. | 01, 02, ..., 12 |
| %M | Minute as a zero-padded decimal number. | 00, 01, ..., 59 |
| %p | Locale's equivalent of either AM or PM. | AM, PM (en_US);<br>am, pm (de_DE) |
| %S | Second as a zero-padded decimal number. | 00, 01, ..., 59 |

# Time format – 2

| Directive | Meaning | Example |
|---|---|---|
| %U | Week number of the year (Sunday as the first day of the week) as a zero padded decimal number. All days in a new year preceding the first Sunday are considered to be in week 0. | 00, 01, ..., 53 |
| %w | 요일에 대한 숫자 표현  0 is Sunday and 6 is Saturday. | 0, 1, ..., 6 |
| %W | Week number of the year (Monday as the first day of the week) as a decimal number. All days in a new year preceding the first Monday are considered to be in week 0. | 00, 01, ..., 53 |
| %x | Locale's appropriate date representation. | 08/16/88 (None);<br>08/16/1988 (en_US);<br>16.08.1988 (de_DE) |
| %X | Locale's appropriate time representation. | 21:30:00 (en_US);<br>21:30:00 (de_DE) |
| %y | 세기 없는 년도 as a zero-padded decimal number. | 00, 01, ..., 99 |
| %Y | Year with century as a decimal number. | 0001, 0002, ..., 2013, 2014, ..., 9998, 9999 |
| %z | UTC offset in the form +HHMM or -HHMM (empty string if the the object is naive). | (empty), +0000, -0400, +1030 |
| %Z | Time zone name (empty string if the object is naive). | (empty), UTC, EST, CST |
| %% | A literal '%' character. | % |

# DATETIME MODULE

# datetime Module class

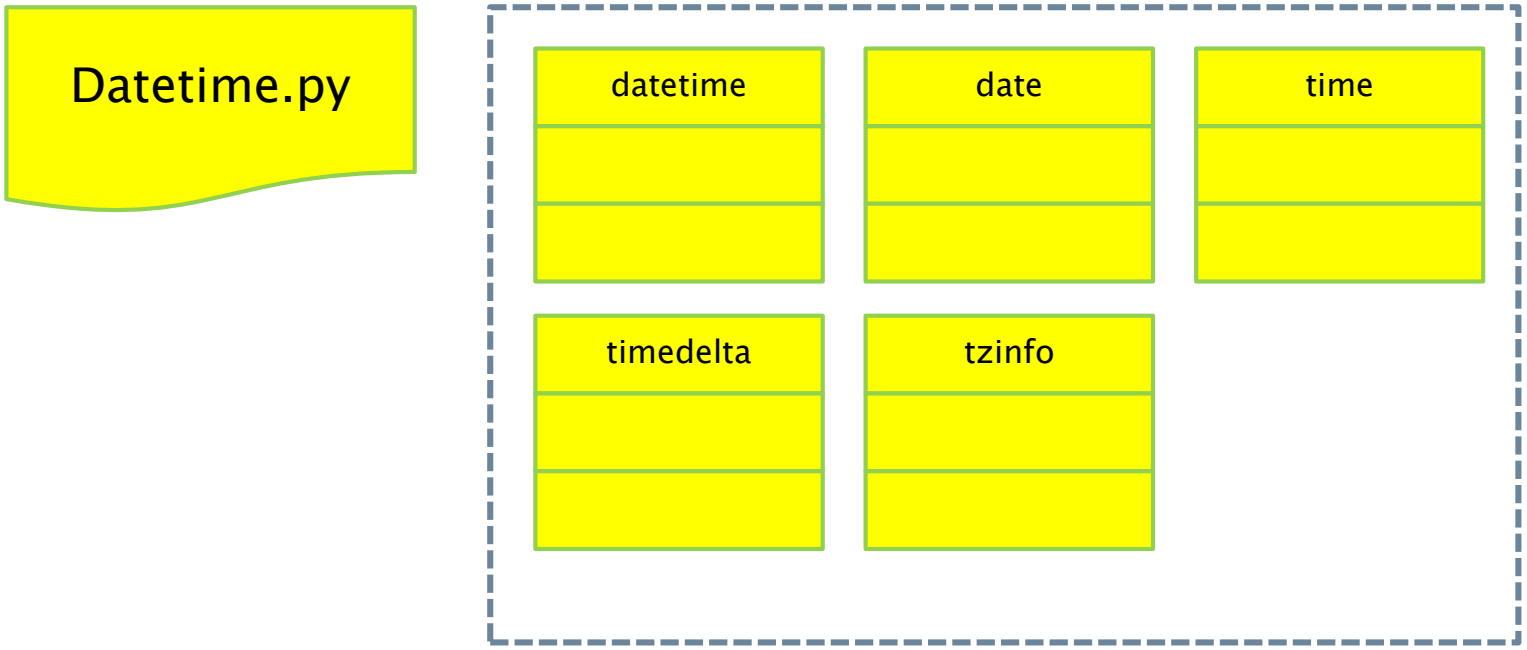
# datetime Module class

## 주요 클래스

| class | Description |
|---|---|
| datetime.date | 일반적인 날짜에 대한 클래스이고(Gregorian calendar 표시) 속성으로 year, month, day를 가짐 |
| datetime.time | 시간에 대한 클래스이고 속성으로 hour, minute, second, microsecond, and tzinfo 가짐 |
| datetime.datetime | date 와 time 클래스 조합을 나타내는 클래스이고 속성은 year, month, day, hour, minute, second, microsecond, and tzinfo를 가짐 |
| datetime.timedelta | 두개의 date, time, or datetime 클래스의 차를 표현하며 days, seconds 속성을 가짐 |
| datetime.tzinfo | 타입존 정보 클래스이며 time고 datetime 클래스 내의 속성으로 처리 |

# 모듈 구조

총 5개의 내부 클래스 구성



# DATETIME .DATE CLASS

# datetime.date

# 클래스 구조

## 3개의 변수와 클래스/인스턴스 메소드로 구성

| date | |
|---|---|
| @classvariable<br>min : datetime.date<br>max: datetime.date<br>Resolution : datetime.timedelta | @instancevariable<br>year : int<br>month : int<br>day : int |
| @classmethod<br>today() : datetime.date<br>fromtimestamp(timestamp:float) : datetime.date<br>fromordinal(ordinal:int) : datetime.date<br><br>@instancemethod<br>weekday() : int<br>isoweekday() : int<br>strftime(format: str) : str<br>isoformat() : str<br>…… | |

# Class/instance variable

| variable | Description |
|---|---|
| date.min | 1년 1월 1일을 나타내는 date(1, 1, 1). |
| date.max | 9999년12월31일을 나타내는 date(9999, 12, 31). |
| date.resolution | <type 'datetime.timedelta'>이며 최소값을 가지고 있음<br>Days: 1 day, seconds : 0:00:00 |
| date.year | Between MINYEAR and MAXYEAR inclusive. |
| date.month | Between 1 and 12 inclusive. |
| date.day | Between 1 and the number of days in the given month of the given year. |

```
import datetime

tod = datetime.date.today()
print(tod.min)
print(tod.max)
print(tod.year, tod.month, tod.day)
```

```
0001-01-01
9999-12-31
2017 1 2
```

# today

| Method | Description |
| --- | --- |
| date.today() | 현재 로컬 date 를 리턴함 즉 , date.fromtimestamp(time.time())와 동등한 결과를 나타남 |

```
import datetime

tod = datetime.date.today()
todate = datetime.datetime.today()
print(type(tod), tod)

print(type(todate), todate)
```

```
<class 'datetime.date'> 2017-01-02
<class 'datetime.datetime'> 2017-01-02 12:03:42.651516
```

# 요일과 갱신

| Method | Description |
|---|---|
| date.weekday() | 월요일(0)부터 일요일(6)까지 표시 |
| date.isoweekday() | 월요일(1)부터 일요일(7)까지 표시 |
| date.replace(*year*, *month*, *day*) | 파라미터를 받아서 수정하면 별도의 date 인스턴스 생김 |
| date.timetuple() | 기존 구조를 time.struct_time( time.localtime()) 로 처리 |

```
import datetime

tod = datetime.date.today()
print(tod.weekday())
print(tod.isoweekday())
print(tod.timetuple())
print(tod.replace(2020,3,30))
```

```
0
1
time.struct_time(tm_year=2017, tm_mon=1, tm_mday=2, t
2020-03-30
```

# 줄리안데이

| Method | Description |
| --- | --- |
| date.toordinal() | 1년 1월 1일부터 date 객체의 날짜까지의 줄리안 데이트를 구함 |
| date.fromordinal(*ordinal*) | 1년 1월 1일 부터 ordinal에 들어온 값을 가지고 일수별로 해당일수 산출 |

```
import datetime

d1 = datetime.date.today()
print(d1.toordinal())

print(d1.fromordinal(736331))
```

```
736331
2017-01-02
```

# Format 처리

| Method | Description |
|---|---|
| date.strftime(*format*) | format에 “%y %m %d” 또는 “%Y %m %d”로 처리하면 년도 월 일로 출력, 연도는 대소문자에 따라 전체 YYYY[대문자] 또는 yy[소문자] 로 표시 |
| date.isoformat() | ISO 8601 format, ‘YYYY-MM-DD’.<br>처리 방법은 , date(2002, 12, 4).isoformat() == '2002-12-04'. |

```
import datetime

cdatetime = datetime.date.today()
cdatetime.strftime("%d %b %Y")

print(type(cdatetime), cdatetime)

dt = cdatetime.isoformat()
print(type(dt),dt)
```

```
<class 'datetime.date'> 2017-01-02
<class 'str'> 2017-01-02
```

# Timestamp에 맞춰 변환

| variable | Description |
| --- | --- |
| date.fromtimestamp(*timestamp*) | POSIX timestamp을 가지고 date 타입으로 전환 예를 들면 time.time() 이 리턴결과는 posix timestamp를 가지고 date 타입으로 리턴함 |

```
import datetime
import time

cdatetime = datetime.date.today()
print(type(cdatetime), cdatetime)

s = cdatetime.fromtimestamp(time.time())
print(type(s), s)
```

```
<class 'datetime.date'> 2017-01-02
<class 'datetime.date'> 2017-01-02
```

# Operator

사칙(+,-)을 하면 datatime.date - datetime.date =timedelta로
datatime.date+timedelta=datatime.date
, 논리연산 을 하면 논리값

```
import datetime

d1 = datetime.date(2016,2,3)
d2 = datetime.date.today()
dd = d2-d1
print(type(dd), dd)

print(d1 > d2)
print(d1 < d2)
d3 = d1 + dd
print(type(d3), d3)
```

```
<class 'datetime.timedelta'> 334 days, 0:00:00
False
True
<class 'datetime.date'> 2017-01-02
```

# Julian 계산 후 odianal 변환

# 예시

## 줄리안을 구해고 다시 원래 날짜를 전환하기

```
import datetime

#줄리안 데이트 구하기
def cal_julian(year,month,day) :
    start = datetime.date(year,month,day)
    print(' start day : ', start)
    end   = datetime.date.today()
    print( ' end   day : ', end)

    time_delta  = end - start
    print( ' julian day : ', time_delta.days)

    return (start,time_delta.days)
#시작일로부터 현재일자 변환
def cal_ordinal(start) :
    x, y = cal_julian(1,1,1)
    ordinal = y  +1
    print( " today :", datetime.date.fromordinal(ordinal))


if __name__ == "__main__" :
    start,julian_day = cal_julian(2015,2,3)
    cal_ordinal(start)
```

```
start day :  2015-02-03
end   day :  2017-01-02
julian day :  699
start day :  0001-01-01
end   day :  2017-01-02
julian day :  736330
today : 2017-01-02
```

# DATETIME . TIME CLASS

# Datetime.time

# 클래스 구조

| time | | |
|---|---|---|
| @classvariable<br>min : datetime.time<br>max: datetime.time<br>resolution : datetime.timedelta | @instancevariable<br>hour :int<br>munite : int<br>second : int | microsecond : int<br>tzinfo ; dateime.tzinfo |
| @instancemethod<br>replace([ hour[, minute[, second[, microsecond[, tzinfo]]]]])<br>isoformat()<br>strftime(format)<br>utcoffset()<br>tdst()<br>tzname() | | |

# Class/instance variable

```
import datetime
loc = datetime.time(22,10,5)
print(loc)
print(loc.min, loc.max)
print(loc.hour)
print(loc.minute)
print(loc.second)
```

```
22:10:05
00:00:00 23:59:59.999999
22
10
5
```

| variable | Description |
|---|---|
| time.min | time(0, 0, 0, 0). |
| time.max | time(23, 59, 59, 999999) |
| time.resolution | timedelta(microseconds=1) |
| time.hour | range(24). |
| time.minute | range(60) |
| time.second | range(60) |
| time.microsecond | range(1000000). |
| time.tzinfo | datetime.tzinfo |

# instance method

## 주요 메소드

| Method | Description |
|---|---|
| time.replace([ *hour*[, *minute*[, *second*[, *microsecond*[, *tzinfo*]]]]]) | 기존 datetime을 갱신한 새로운 datetime 인스터스를 생성 |
| time.isoformat() | HH:MM:SS.mmmmmm, if self.microsecond is 0, HH:MM:SS |
| time.strftime(*format*) | format 정의에 따라 출력 |
| time.utcoffset() | If tzinfo is None, returns None, else returns self.tzinfo.utcoffset(None), and raises an exception if the latter doesn't return None or a timedelta object representing a whole number of minutes with magnitude less than one day. |
| time.dst() | If tzinfo is None, returns None, else returns self.tzinfo.dst(None), and raises an exception if the latter doesn't return None, or a timedelta object representing a whole number of minutes with magnitude less than one day. |
| time.tzname() | If tzinfo is None, returns None, else returns self.tzinfo.tzname(None), or raises an exception if the latter doesn't return None or a string object. |

# instance method 처리 예시

Time처리시 tzinfo를 subclass 구현한 후 처리하기

```
from datetime import time, tzinfo,timedelta

#tzinfo 추상클래스를 현행화
class GMT1(tzinfo):
    def utcoffset(self, dt):
        return timedelta(hours=1)
    def dst(self, dt):
        return timedelta(0)
    def tzname(self,dt):
        return "Europe/Prague"

#시간생성  tzinfo에 객체 생성
t = time(12, 10, 30, tzinfo=GMT1())
print( "time instance    : ",t)
print( "time iso format : ",t.isoformat())
print( "time dst          : ", t.dst())
print( 'time zone         : ',t.tzname())
print( "time strftime    : ",t.strftime("%H:%M:%S %Z"))
print( 'The {} is {:%H:%M}.'.format("time", t))
```

```
time instance    :  12:10:30+01:00
time iso format :  12:10:30+01:00
time dst          :  0:00:00
time zone         :  Europe/Prague
time strftime    :  12:10:30 Europe/Prague
The time is 12:10.
```

# DATETIME . DATETIME CLASS

# datetime.datetime

# 클래스 구조

| datetime | | |
|---|---|---|
| @classvariable<br>min : datetime.date<br>max: datetime.date<br>resolution : datetime.timedelta | @instancevariable<br>year : int<br>month : int<br>day : int<br>hour :int | munite : int<br>second : int<br>microsecond : int<br>tzinfo ; dateime.tzinfo |
| @classmethod<br>today() : datetime.datetime<br>now(tz) : datetime.datetime<br>utcnow() : datetime.datetime<br>fromtimestamp(timestamp[, tz]) : datetime.datetime<br>fromordinal(ordinal) : datetime.datetime<br>combine(date, time) : datetime.datetime<br>strptime(date_string, format) : datetime.datetime<br><br>@instancemethod<br>date()<br>time()<br>timetz()<br><br>strftime(format: str) : str<br>isoformat() : str<br>astimezone(tz:str) : tzinfo | | |

.....

# Class variable

| variable | Description |
| --- | --- |
| datetime.min | 1년 1월 1일을 나타내는 date(1, 1, 1). |
| datetime.max | 9999년12월31일을 나타내는 date(9999, 12, 31). |
| datetime.resolution | <type 'datetime.timedelta'>이며 최소값을 가지고 있음<br>Days: 1 day, seconds : 0:00:00 |

```
import datetime

print(datetime.datetime.min)
print(datetime.datetime.max)
print(datetime.datetime.resolution)
```

```
0001-01-01 00:00:00
9999-12-31 23:59:59.999999
0:00:00.000001
```

# instance variable

```
import datetime

d1 = datetime.datetime.today()
print(d1.year)
print(d1.month)
print(d1.day)
print(d1.hour)
print(d1.minute)
print(d1.second)
print(d1.microsecond)
print(d1.tzinfo)
```

```
2017
1
2
13
11
23
855215
None
```

| variable | Description |
|---|---|
| datetime.year | Between MINYEAR and MAXYEAR inclusive. |
| datetime.month | Between 1 and 12 inclusive. |
| datetime.day | Between 1 and the number of days in the given month of the given year. |
| datetime.hour | range(24). |
| datetime.minute | range(60) |
| datetime.second | range(60) |
| datetime.microsecond | range(1000000). |
| datetime.tzinfo | datetime.tzinfo |

# Now/today

| Method | Description |
| --- | --- |
| datetime.today() | 현재 날짜를 date와 time을 전부 보여줌 datetime.datetime(년,월,일,시,분,초,마이크로초) |
| datetime.now([*tz*]) | 현재의 타임존에 date와 time을 보여줌. 타임존(tzinfo)이 없을 경우는 datetime.today() 와 동일 |
| datetime.utcnow() | UTC를 기준으로 현재 date와 time을 표시 |

```
import datetime
print(datetime.datetime.now())
print(datetime.datetime.utcnow())
print(datetime.datetime.today())
```

```
2017-01-02 13:07:37.958914
2017-01-02 04:07:37.958914
2017-01-02 13:07:37.958915
```

# Julian date

| Method | Description |
| --- | --- |
| datetime.fromordinal(*ordinal*) | 1년 1월 1일부터 ordinal(일수)를 넣으면 계산해서 datetime.datetime으로 반환, time에 대한 정보는 초기화됨 |
| datetime.toordinal(*ordinal*) | |

```
import datetime

d1 = datetime.datetime.today()
print(d1.toordinal())

print(d1.fromordinal(736331))
```

```
736331
2017-01-02 00:00:00
```

# Datetime 생성

| Method | Description |
| --- | --- |
| datetime.fromtimestamp(*timestamp*[, *tz*]) | Posix timestampe(time.time())을 기준으로 datetime.datetime으로 표시 |
| datetime.combine(*date*, *time*) | datetime.date와 datetime.time 객체를 datetime.datetime으로 조합하여 표시 |

```
import datetime
import time

a = datetime.datetime.fromtimestamp(time.time())
print(a)

b = datetime.datetime.combine(datetime.date.today(), datetime.time(22,22,22))
print(b)
```

```
2017-01-02 14:17:35.142168
2017-01-02 22:22:22
```

# instance method

## 주요 메소드

| Method | Description |
| --- | --- |
| datetime.date() | datetime에서 date로 변환 |
| datetime.time() | datetime에서 time로 변환 |
| datetime.timetz() | datetime에서 time로 변환 time zone 정보가 없을 경우는 datetime.time() 동일 |
| datetime.replace([*year*[, *month*[, *day*[, *hour*[, *minute*[, *second*[, *microsecond*[, *tzinfo*]]]]]]]]) | 기존 datetime을 갱신한 새로운 datetime 인스터스를 생성 |
| datetime.astimezone(*tz*) | 현재 타입존을 가져온다. 타입존에 대해서는 실제 구현하여 사용해야 함<br>>>> now.astimezone(date_example.MyUTCOffsetTimezone()) |
| datetime.utcoffset() | utc와의 차이를 표시. Timedelta로 표시됨 |
| datetime.dst() | 일광시간대가 표시여부 |
| datetime.tzname() | |
| datetime.timetuple() | |

# Formatting 처리

# datetime format 예시

datetime에 대한 format 처리

```
# YYYYmmddHHMMSS 형태의 시간 출력

datetime.today().strftime("%Y%m%d%H%M%S")

# YYYY/mm/dd HH:MM:SS 형태의 시간 출력

datetime.today().strftime("%Y/%m/%d %H:%M:%S")
```

# Formatting 처리

| Method | Description |
| --- | --- |
| datetime.strftime(*format*) | format에 맞춰 출력 |
| datetime.strptime(*date_string*, *format*) | datetime.datetime.strptime("21/11/06 16:30", "%d/%m/%y %H:%M") ➔ datetime.datetime(2006, 11, 21, 16, 30) |

```
import datetime

cdatetime = datetime.datetime.now()
cdatetime.strftime("%d %b %Y %I:%M:%S %p")

print(type(cdatetime), cdatetime)

dt = datetime.datetime.strptime('17 Feb 2009 04:22:11 PM','%d %b %Y %I:%M:%S %p')
print(type(dt),dt)
```

```
<class 'datetime.datetime'> 2017-01-02 13:02:38.156684
<class 'datetime.datetime'> 2009-02-17 16:22:11
```

# UTC/GMT/EST

# UTC/GMT/EST 처리

datetime.datetime 내에서 UTC/GMT/EST를 기준으로 처리

```
from datetime import tzinfo,timedelta
import datetime

class Zone(tzinfo):
    def __init__(self,offset,isdst,name):
        self.offset = offset
        self.isdst = isdst
        self.name = name
    def utcoffset(self, dt):
        return timedelta(hours=self.offset) + self.dst(dt)
    def dst(self, dt):
            return timedelta(hours=1) if self.isdst else timedelta(0)
    def tzname(self,dt):
         return self.name

GMT = Zone(0,False,'GMT')
EST = Zone(-5,False,'EST')

print( datetime.datetime.utcnow().strftime('%m/%d/%Y %H:%M:%S %Z'))
print( datetime.datetime.now(GMT).strftime('%m/%d/%Y %H:%M:%S %Z'))
print( datetime.datetime.now(EST).strftime('%m/%d/%Y %H:%M:%S %Z'))
```

```
01/02/2017 04:44:33
01/02/2017 04:44:33 GMT
01/01/2017 23:44:33 EST
```

# DATETIME . TIMEDELTA CLASS

# Timedelta 생성자

class datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)

- timedelta에 들어갈 수 있는 인자값은 아래와 같다.
  - 1주 : datetime.timedelta(weeks=1)
  - 1일 : datetime.timedelta(days=1)
  - 1시간 : datetime.timedelta(hours=1)
  - 1분 : datetime.timedelta(minutes=1)
  - 1초 : datetime.timedelta(seconds=1)
  - 1밀리초 : datetime.timedelta(milliseconds=1)
  - 1마이크로초 : datetime.timedelta(microseconds=1)
- timedelta로 5시간 30분을 표현하면 datetime.timedelta(hours=5, minutes=30)이라고 하면 된다.

# Timedelta instance 변수

days, seconds,microseconds로 표시되며 read-only

| Attribute | Value |
| --- | --- |
| days | Between -999999999 and 999999999 inclusive |
| seconds | Between 0 and 86399 inclusive |
| microseconds | Between 0 and 999999 inclusive |

# Timedelta 메소드

total_seconds 메서드를 호출하면 초단위로 쉽게 변경할 수 있다.

```
from datetime import timedelta

year = timedelta(days=365)
another_year = timedelta(weeks=40, days=84, hours=23,
                         minutes=50, seconds=600) # adds up to 365 days

print(year.total_seconds())
print(another_year.total_seconds())
print(year == another_year )
```

```
31536000.0
31536000.0
True
```

# Timedelta: 내일 구하기

datatime(오늘) + timedelta(days=1) = datatime(내일)

```
import datetime

now2 = datetime.datetime.now()

#오늘에 하루를 더해서 내일 구하기
tomorrow = now2 + datetime.timedelta(days=1)
print(tomorrow)
```

```
2017-01-03 13:49:43.826767
```

# timedelta : 일자, 시간 등 변경

datatime - datatime = timedelta

```
import datetime

oneDatetime = datetime.datetime.strptime('2015-04-15 00:00:00', '%Y-%m-%d %H:%M:%S')
twoDatetime = datetime.datetime.strptime('2015-04-16 00:00:10', '%Y-%m-%d %H:%M:%S')

result = twoDatetime - oneDatetime
print(result)
print(result.days)
print(result.seconds)
```

```
1 day, 0:00:10
1
10
```

# Utcoffset()에서 timedelta 예시

UTC 시간과 timezone 시간이 차도 timedelta로 표시

```
import datetime
import pytz


tz = pytz.timezone('America/St_Johns')

st_johns_dt = tz.normalize(datetime.datetime.now(tz=pytz.utc))
print( " ameria / st_johns ", st_johns_dt)

print( " utc offset ",st_johns_dt.utcoffset())
st_tmdt = st_johns_dt.utcoffset()
print( int(st_tmdt.total_seconds()/3600) )
print( int((st_tmdt.total_seconds()%3600)/60) )
```

```
 ameria / st_johns  2017-01-02 01:22:14.207167-03:30
 utc offset  -1 day, 20:30:00
-3
30
```

# DATETIME . TZINFO CLASS

# datetime.tzinfo

# datetime.tzinfo

time zone을 나타내는 추상화 클래스

```
import datetime

print(type(datetime.tzinfo))
print(help(datetime.tzinfo))
```

```
<class 'type'>
Help on class tzinfo in module datetime:

class tzinfo(builtins.object)
 |  Abstract base class for time zone info objects.
 |
```

# tzinfo 클래스

tzinfo 는 타임존에 대한 정보를 가지는 클래스

```
from datetime import tzinfo

print(type(tzinfo))
print(dir(tzinfo))

i = tzinfo()
print(i)
```

```
<class 'type'>
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format
 '__init__', '__le__', '__lt__', '__ne__', '__new__', '__reduce__', '_
_str__', '__subclasshook__', 'dst', 'fromutc', 'tzname', 'utcoffset']
<datetime.tzinfo object at 0x000000000045F4C0>
```

tzinfo.utcoffset(*self*, *dt*)
tzinfo.dst(*self*, *dt*)
tzinfo.tzname(*self*, *dt*)
tzinfo.fromutc(*self*, *dt*)

# tzinfo : Subclass 활용 예시

Tzinfo 클래스는 재정의해서 사용해야 함

```
from datetime import tzinfo, timedelta, datetime
from pytz import UTC


class MyUTCOffsetTimezone (tzinfo):

    def __init__(self, offset=19800, name=None):
        self.offset = timedelta(seconds=offset)
        self.name = name or self.__class__.__name__

    def utcoffset(self, dt):
        return self.offset

    def tzname(self, dt):
        return self.name

    def dst(self, dt):
        return timedelta(0)

now = datetime.now(tz=UTC)
print(now)

print(now.astimezone(MyUTCOffsetTimezone()))

print(datetime.now(MyUTCOffsetTimezone()))
```

```
2017-01-02 02:47:38.481446+00:00
2017-01-02 08:17:38.481446+05:30
2017-01-02 08:17:38.482446+05:30
```