파이썬
코딩의 기술

CH 44 ~ 48

CH 44. copyreg로 pickle을 신뢰할 수 있게 만들자

pickle이란?

파이썬 객체를 바이트 스트림으로 직렬화하거나 바이트를 객체로 역직렬화 하는데 사용한다.

```
import pickle


class GameState(object):
    def __init__(self):
        self.level = 0
        self.lives = 4

state = GameState()
state.level += 1  # Player beat a level
state.lives -= 1  # Player had to try again

serialized = pickle.dumps(state)
state_after = pickle.loads(serialized)
print(state_after.__dict__)
```

```
{'lives': 3, 'level': 1}
```

pickle 사용시 문제가 될 수 있는 상황.1

1) pickle을 통해 클래스 객체를 파일로 저장

2) 클래스에 변수가 추가

3) 이미 저장된 객체가 unpickle될 때의 상황

```
class GameState(object):
    def __init__(self):
        self.level = 0
        self.lives = 4

state = GameState()
state.level += 1  # Player beat a level
state.lives -= 1  # Player had to try again
```

```
state_path = 'game_state.bin'
with open(state_path, 'wb') as f:
    pickle.dump(state, f)

with open(state_path, 'rb') as f:
    state_after = pickle.load(f)
print(state_after.__dict__)
```

Version Up

```
class GameState(object):
    def __init__(self):
        self.level = 0
        self.lives = 4
        self.points = 0
```

game_state.bin

```
with open(state_path, 'rb') as f:
    state_after = pickle.load(f)
print(state_after.__dict__)
```

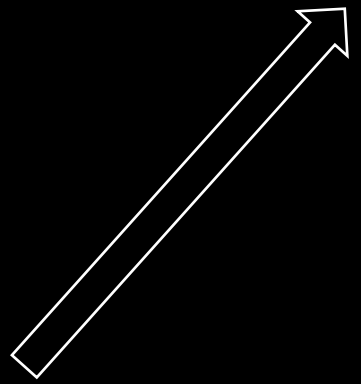
```
{'level': 1, 'lives': 3}
```

copyreg 이용하기

```python
import pickle
import copyreg


class GameState(object):
    def __init__(self, level=0, lives=4):
        self.level = level
        self.lives = lives


def pickle_game_state(game_state):
    kwargs = game_state.__dict__
    return unpickle_game_state, (kwargs,)


def unpickle_game_state(kwargs):
    return GameState(**kwargs)

copyreg.pickle(GameState, pickle_game_state)

state = GameState()
state.level += 1  # Player beat a level
state.lives -= 1  # Player had to try again

with open('game_state.bin', 'wb') as f:
    pickle.dump(state, f)

print(state.__dict__)
```

game_state.bin

```
└c__main__ounpickle_game_stateoq }q┌(X|    levelq┐K┌X|    livesq└K└u웹┘Rq|.
```



```python
import pickle


class GameState(object):
    def __init__(self, level=0, lives=4, points=0):
        self.level = level
        self.lives = lives
        self.point = points


def unpickle_game_state(kwargs):
    return GameState(**kwargs)


with open('game_state.bin', 'rb') as f:
    state_after = pickle.load(f)

print(state_after.__dict__)
```

```
{'point': 0, 'level': 1, 'lives': 3}
```

copyreg 등록 전, 후 결과 비교

```python
import pickle
import copyreg


class GameState(object):
    def __init__(self, level=0, lives=4):
        self.level = level
        self.lives = lives


def pickle_game_state(game_state):
    kwargs = game_state.__dict__
    return unpickle_game_state, (kwargs,)


def unpickle_game_state(kwargs):
    return GameState(**kwargs)

copyreg.pickle(GameState, pickle_game_state)

state = GameState()
state.level += 1  # Player beat a level
state.lives -= 1  # Player had to try again

with open('game_state.bin', 'wb') as f:
    pickle.dump(state, f)

print(state.__dict__)
```

```python
serialize = pickle.dumps(state)
print(serialize)
```

```
b'\x80\x03c__main__\nGameState\nq\x00)\x81q\x01}q\x02(X\x05\x00\x00\x00levelq\x03K\x01X\x05\x00\x00\x00livesq\x04K\x03ub.'
```

```python
copyreg.pickle(GameState, pickle_game_state)
serialize = pickle.dumps(state)
print(serialize)
```

```
b'\x80\x03c__main__\nunpickle_game_state\nq\x00}q\x01(X\x05\x00\x00\x00levelq\x02K\x01X\x05\x00\x00\x00livesq\x03K\x03u\x85q\x04Rq\x05.'
```

pickle 사용시 문제가 될 수 있는 상황.2

1) pickle을 통해 클래스 객체를 파일로 저장

2) 클래스에 변수가 삭제

3) 이미 저장된 객체가 unpickle될 때의 상황

```
class GameState(object):
    def __init__(self, level=0, points=0):
        self.level = level
        self.point = points


def unpickle_game_state(kwargs):
    return GameState(**kwargs)


with open('game_state.bin', 'rb') as f:
    state_after = pickle.load(f)

print(state_after.__dict__)
```

```
TypeError: __init__() got an unexpected keyword argument 'lives'
```

```
import pickle
import copyreg


class GameState(object):
    def __init__(self, level=0, points=0):
        self.level = level
        self.point = points


def pickle_game_state(game_state):
    kwargs = game_state.__dict__
    kwargs['version'] = 2
    return unpickle_game_state, (kwargs,)


def unpickle_game_state(kwargs):
    version = kwargs.pop('version', 1)
    if version == 1:
        kwargs.pop('lives')
    return GameState(**kwargs)

copyreg.pickle(GameState, pickle_game_state)

with open('game_state.bin', 'rb') as f:
    state_after = pickle.load(f)

print(state_after.__dict__)
```

pickle 사용시 문제가 될 수 있는 상황.3

pickle, unpickle이 적용될 클래스의 이름 변경

GameState -> BetterGameState 로 이름 변경

```
serialize = pickle.dumps(state)
print(serialize)
```

```
b'₩x80₩x03c__main__₩nGameState₩nq₩x00)₩x81q₩x01}q₩x
```

```
def pickle_game_state(game_state):
    kwargs = game_state.__dict__
    return unpickle_game_state, (kwargs,)
```

```
def unpickle_game_state(kwargs):
    return BetterGameState(**kwargs)
```

```
copyreg.pickle(GameState, pickle_game_state)
serialize = pickle.dumps(state)
print(serialize)
```

```
b'₩x80₩x03c__main__₩nunpickle_game_state₩nq₩x00}q₩x01(
```

CH 45. 지역 시간은 time이 아닌 datetime으로 표현하자

time모듈 사용하기

UTC → 지역 시간

```
from time import localtime, strftime

now = 1407694710
local_tuple = localtime(now)
time_format = '%Y-%m-%d %H:%M:%S'
time_str = strftime(time_format, local_tuple)
print(time_str)
```

```
2014-08-11 03:18:30
```

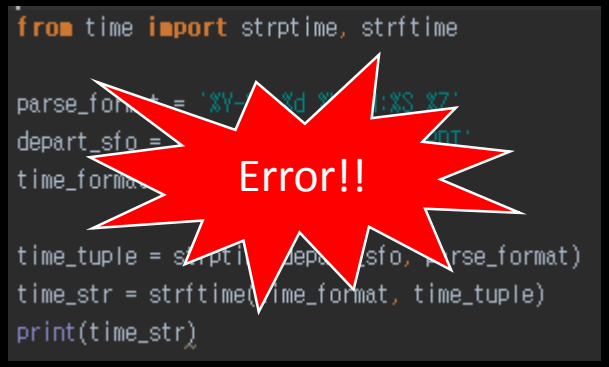
UTC ← 지역 시간

```
from time import mktime, strptime

time_str = '2014-08-11 03:18:30'
time_format = '%Y-%m-%d %H:%M:%S'
time_tuple = strptime(time_str, time_format)
utc_now = mktime(time_tuple)
print(utc_now)
```

```
1407694710.0
```

time모듈 사용하기

지역 시간 → 다른 지역 시간




```
ValueError: time data '2014-05-01 15:45:16 PDT' does not match format '%Y-%m-%d %H:%M:%S %Z'
```

책에서는 위 코드가 정상 동작 한다 하지만 Windows에서는 안된다.
왜냐하면 time모듈은 운영체제 플랫폼에 의존적이기 때문이다.(GMT, UTC는 된다.)

datetime모듈 사용하기

UTC → 지역 시간

```
from datetime import datetime, timezone

now = datetime(2014, 8, 10, 18, 18, 30)
now_utc = now.replace(tzinfo=timezone.utc)
now_local = now_utc.astimezone()
print(now)
print(now_utc)
print(now_local)
```

```
2014-08-10 18:18:30
2014-08-10 18:18:30+00:00
2014-08-11 03:18:30+09:00
```

UTC ← 지역 시간

```
from datetime import datetime
from time import mktime

time_str = '2014-08-10 11:18:30'
time_format = '%Y-%m-%d %H:%M:%S'
now = datetime.strptime(time_str, time_format)
time_tuple = now.timetuple()
utc_now = mktime(time_tuple)
print(now)
print(time_tuple)
print(utc_now)
```

```
2014-08-10 11:18:30
time.struct_time(tm_year=2014, tm_mon=8, tm_mday=10, tm_hour=11, tm_min=18, tm_sec=30, tm_wday=6, tm_yday=222, tm_isdst=-1)
1407637110.0
```

datetime, pytz모듈 사용하기

지역 시간 → 다른 모든 지역 시간으로 변경하기 위해서는 pytz를 사용해서 변경하자.
(datetime만을 사용하는 것은 일부 지원되지 않는 시간대가 있다.)

pytz : 모든 시간대에 대한 정의를 담은 전체 데이터 베이스

```
from datetime import datetime
import pytz

arrival_nyc = '2014-05-01 23:33:24'
time_format = '%Y-%m-%d %H:%M:%S'
nyc_time_tuple = datetime.strptime(arrival_nyc, time_format)
print(nyc_time_tuple)

eastern = pytz.timezone('US/Eastern')
nyc_dt = eastern.localize(nyc_time_tuple)
print(nyc_dt)

utc_dt = pytz.utc.normalize(nyc_dt.astimezone(pytz.utc))
print(utc_dt)

pacific = pytz.timezone('US/Pacific')
sf_dt = pacific.normalize(utc_dt.astimezone(pacific))
print(sf_dt)
```

이렇게 해야 운영체제와 상관없이
모든 환경에서 동일하게 동작한다.

```
2014-05-01 23:33:24
2014-05-01 23:33:24-04:00
2014-05-02 03:33:24+00:00
2014-05-01 20:33:24-07:00
```

CH 46. 내장 알고리즘과 자료 구조를 사용하자

더블 엔디드 큐(double-ended queue)

collections 모듈의 deque클래스

- 처음과 끝에서 아이템을 삽입, 삭제 할 때 항상 일정한 시간이 걸리는 연산을 제공
- 선입선출(FIFO)큐를 만들 때 이상적이다.
- 내장타입 list도 같은 기능을 할 수 있지만 리스트의 시작 부분에서 아이템을 삽입, 삭제하는 연산에는 선형적 시간이 걸리므로 deque의 일정한 시간보다 훨씬 느리다.

```
from collections import deque

fifo = deque()
fifo.append(1)      # Producer
fifo.append(2)
x = fifo.popleft()  # Consumer
print(x)
```

```
1
```

정렬된 딕셔너리

- 표준 딕셔너리는 정렬되어 있지 않다. 즉, 순회할 때 아이템의 순서가 바뀔 수 있다.
- 이는 딕셔너리의 빠른 해시 테이블을 구현하는 방식이 만들어낸 뜻밖의 부작용이다.
- collections 모듈의 OrderedDict는 키가 삽입된 순서를 유지하는 특별한 딕셔너리 타입이다.

```
a = {}
a['foo'] = 1
a['bar'] = 2

from random import randint

# Randomly populate 'b' to cause hash conflicts
while True:
    z = randint(99, 1013)
    b = {}
    for i in range(z):
        b[i] = i
    b['foo'] = 1
    b['bar'] = 2
    for i in range(z):
        del b[i]

    if str(b) != str(a):
        break

print(a)
print(b)
print('Equal?', a == b)
```

```
{'bar': 2, 'foo': 1}
{'foo': 1, 'bar': 2}
Equal? True
```

```
{'foo': 1, 'bar': 2}
{'bar': 2, 'foo': 1}
Equal? True
```

```
from collections import OrderedDict
a = OrderedDict()
a['foo'] = 1
a['bar'] = 2

b = OrderedDict()
b['foo'] = 'red'
b['bar'] = 'blue'

for value1, value2 in zip(a.values(), b.values()):
    print(value1, value2)
```

```
1 red
2 blue
```

기본 딕셔너리

딕셔너리를 사용할 때 한가지 문제는 어떤 키가 이미 존재한다고 가정할 수 없는 점이다.

```
stats = {}
key = 'my_counter'
if key not in stats:
    stats[key] = 0
stats[key] += 1
print(stats)
```

```
{'my_counter': 1}
```

collections 모듈의 defaultdict는 키가 존재하지 않으면 자동으로 기본값을 저장하도록 한다.
하는 일은 키가 없을 때 기본값을 반환 할 함수만 넘겨주면 된다.

```
print(int())
```

```
0
```

```
from collections import defaultdict
stats = defaultdict(int)
stats['my_counter'] += 1
print(dict(stats))
```

```
{'my_counter': 1}
```

힙 큐

힙은 우선순위 큐를 유지한다.
heapq모듈은 표준 list타입으로 힙을 생성하는 heappush, heappop, nsmallest 함수를 제공
sort 메서드를 호출해도 힙의 불변성 유지

```
from heapq import *
a = []
heappush(a, 5)
heappush(a, 3)
heappush(a, 7)
heappush(a, 4)

print(heappop(a), heappop(a), heappop(a), heappop(a))
```

```
3 4 5 7
```

```
from heapq import *
a = []
heappush(a, 5)
heappush(a, 3)
heappush(a, 7)
heappush(a, 4)

assert a[0] == nsmallest(1, a)[0] == 3
```

```
from heapq import *
a = []
heappush(a, 5)
heappush(a, 3)
heappush(a, 7)
heappush(a, 4)

print('Before:', a)
a.sort()
print('After: ', a)
```

```
Before: [3, 4, 7, 5]
After:  [3, 4, 5, 7]
```

heapq 연산에 걸리는 시간은 리스트의 길이에 비례하여 로그 형태로 증가한다.
리스트로 같은 동작을 하려면 시간이 선형적으로 증가한다.

바이섹션

- list에서 아이템 검색을 위해 index()를 호출할 때 리스트 길이에 비례한 시간이 걸린다.
- bisect모듈의 bisect_left()는 정렬된 아이템 시퀀스를 대상으로 효율적인 바이너리 검색을 제공한다.

```
import time

start = time.time()
x = list(range(10**6))
i = x.index(991234)
end = time.time() - start
print(end)
```

```
0.07058882713317871
```

```
import time
from bisect import bisect_left

start = time.time()
x = list(range(10**6))
i = bisect_left(x, 991234)
end = time.time() - start
print(end)
```

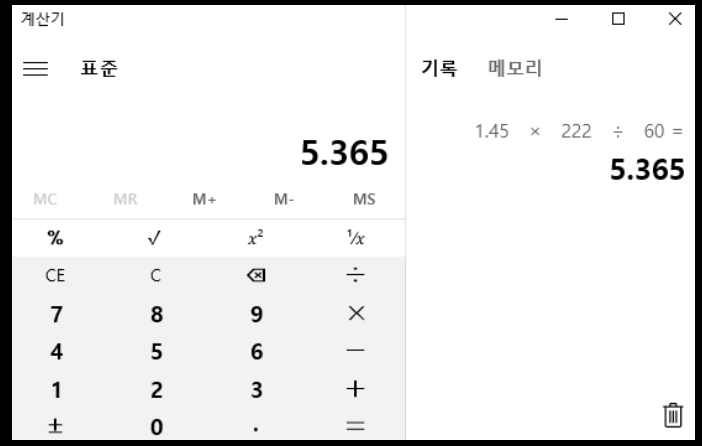
```
0.0390467643737793
```

CH 47. 정밀도가 중요할 때는 Decimal을 사용하자

Decimal을 사용하자

```
rate = 1.45
seconds = 3*60 + 42
cost = rate * seconds / 60
print(cost)

print(round(cost, 2))
print('-------------------------')
rate = 0.05
seconds = 5
cost = rate * seconds / 60
print(cost)

print(round(cost, 2))
```

```
5.364999999999999
5.36
-------------------------
0.004166666666666667
0.0
```




```
from decimal import Decimal
from decimal import ROUND_UP

rate = Decimal('1.45')
seconds = Decimal('222')  # 3*60 + 42
cost = rate * seconds / Decimal('60')
print(cost)

rounded = cost.quantize(Decimal('0.01'), rounding=ROUND_UP)
print(rounded)
print('-------------------------')
rate = Decimal('0.05')
seconds = Decimal('5')
cost = rate * seconds / Decimal('60')
print(cost)

rounded = cost.quantize(Decimal('0.01'), rounding=ROUND_UP)
print(rounded)
```

```
5.365
5.37
-------------------------
0.004166666666666666666666666667
0.01
```

CH 48. 커뮤니티에서 만든 모듈을 어디서 찾아야 하는지 알아두자

파이썬 패키지 인덱스(PyPI) https://pypi.python.org/pypi

pip은 파이썬 3.4 이후에는 자동으로 설치된다.

```
C:₩>pip3 install pytz
```

- pip3는 패키지의 파이썬3 버전을 설치한다.
- pip는 패키지의 파이썬2 버전을 설치한다.

PyCharm이 참 좋다.