1.18 시퀀스 요소에 이름 매핑

[문제] 리스트나 튜플의 위피로 요소에 접근하는 코드가 있음. 때때로 가독성이 떨어짐. 위치에 의존하는 코드의 구조도 이름으로 접근가능하도록 수정하고 싶음

■ collections.namedtuple() 사용 : 일반적인 튜플 객체를 사용하는 것에 비해 크지 않은 오버헤드로 이 기능을 구현. 타입이름& 포함해야할 필드를 전달하면 인스턴스화 가능한 클래스를 반환함. 필드의 값을 전달하는 식으로 사용 가능

```
from collections import namedtuple
Subscriber = namedtuple('Subscriber',['addr','joined'])
sub = Subscriber('jonesy@example.com','2012-10-19')
print(sub)          #>>> Subscriber(addr='jonesy@example.com', joined='2012-10-19')
print(sub.addr)     #>>> jonesy@example.com
print(sub.joined)   #>>> 2012-10-19
```

namedtuple의 인스턴스는 일반적인 클래스 인스턴스와 비슷해보이지만 튜플과 교환이 가능하고 인덱싱이나 언패킹 같은 일반적인 기능을 모두 지원함

```
len(sub)
addr, joined = sub
print(addr)     #>>>jonesy@example.com
print(joined)   #>>>2012-10-19
```

named tuple은 주료 요소의 위치를 기반으로 구현되어 있는 코드를 분리함
—> 데이터베이스로부터 거대한 튜플 리스트를 받고 요소의 위치로 접근할 수 있는 코드가 있을 때 (ex) 테이블에 새로운 열 추가) 문제가 생김 but, 변환된 튜플을 네임드튜플로 변환하면 문제 안 생김

ex1) 일반적인 튜플을 사용하는 경우

```
def compute_cost(records):
    total =0.0
    for rec in records:
        total += rec[1] * rec[2]
    return total
```

—>가독성 x, 일반적인 구조형에 크게 의존

ex2) 네임드 튜플

```
from collections import namedtuple
Stock = namedtuple('Stock', ['name','shares','price'])
def compute_cost(records):
    total = 0.0
    for rec in records:
        s = Stock(*rec) #* 다시 보기
        total += s.shares * s.price
    return total
```

—>records 시퀀스에 이미 인스턴스가 포함되어 있어서 Stock네임드 튜플로 명시적인 변환을 하지 않아도 됨

namedtuple은 저장공간을 더 필요로 하는 딕셔너리 대신 사용 가능
딕셔너리를 포함한 방대한 자료 구조를 구상하고 있다면 namedtuple을 사용하는 것이 효율적. but, 딕셔너리와는 다르게 네임드 튜플은 수정불가

```
s= Stock('ACME',100,123.45)
print(s) #S>>> tock(name='ACME', shares=100, price=123.45)
s.share = 75 >>> 오류남
```

속성 수정 시 namedtuple인스턴스의 _replace()메소드 사용 : 지정한 값을 치환하여 완전히 새로운 네임드 튜플을 만듦

```
s = s._replace(shares=75)
print(s) #>>> Stock(name='ACME', shares=75, price=123.45)
```

_replace()메소드를 사용하면 옵션이나 빈 필드를 가진 네임드 튜플을 간단히 만들수 있음.
HOW? 기본 값을 가진 프로토타입 튜플 생성 -> _replace()로 치환된 값을 새로운 인스턴스 생성

```
from collections import namedtuple
Stock = namedtuple('Stock', ['name','share','price','date','time'])

#프로토타입 인스턴스 생성 #프로토타입이 뭐시당가 # 대충 타입 지정해주는 거 같음
stock_prototype = Stock('',0,0.0,None,None)

#딕셔너리를 Stock으로 변환하는 함수
def dict_to_stock(s):
    return stock_prototype._replace(**s)
```

[예시]

```
a = {'name':'ACME', 'shares':100, 'price' :123.45}
print ( dict_to_stock(a))        #Stock(name='ACME', shares=100, price=123.45, date=None, time=None)
b = {'name':'ACME', 'shares':100, 'price' : 123.45, 'date':'12/17/2012'}
print(dict_to_stock(b))          #Stock(name='ACME', shares=100, price=123.45, date='12/17/2012',
time=None)
```

여러 인스턴스 요소를 빈번히 수정해야하는 자료구조를 만들 경우 네임드 튜플은 노노
__slot__을 사용해서 클래스를 정의하는 것을 고려(8.4)

1.19 데이터를 변환하면서 줄이기 (뭐라하는지 하나도 모르겠음)

- 감소함수(ex) sum(), min(), max())) 실행 시 먼저 데이터 변환과 필터링을 해야함
- > 생성자 표현식 사용

```
ex)정사각형 넓이의 합 계산
nums = [1,2,3,4,5]
s = sum(x * x for x in nums)
print(s) # >>> 55
```

대안으로 다음과 같은 코드도 가능

```
#디렉토리에 또다른 .py파일이 있는지 살펴봄
import os
files = os.listdir('dirname')
if any(name.endswith('.py') for name in files):
    print( 'there be python')
else:
    print('sorry. no python')

#튜플을 csv로 출력
s = ('ACME',50,123.45)
print( ','.join(str(x) for x in s))

#자료 구조의 필드를 줄인다
portfolio = [
    {'name':'GOOD', 'shares':50},
    {'name':'YHOO', 'shares':75},
    {'name':'AOL', 'shares':20},
    {'name':'SCOX', 'shares':65}
]
min_shares = min( s['shares'] for s in portfolio ) 뭔소리여
```

함수에 인자로 전달된 생성자 표현식의 문법적인 측면을 보여줌(즉, 반복적인 괄호를 할 필요는 없음)

아래 코드는 동일함
s = sum((x* x for x in nums))
s = sum(x * x for x in noms)

생성자 인자를 사용하면 임시 리스트를 만드는 것보다 더 효율적이고 코드가 간결한 경우가 많음
ex) 생성자 표현식을 사용하지 않을 경우 다음과 같은 코드를 작성해야 함
noms = [1,2,3,4,5]
s = sum([x*x for x in nums])
위 코드도 동작하지만 추가적인 리스트를 생성해야 한다는 번거로움이 있음.

작은 리스트는 크게 문제가 되지 않지만 nums의 크기가 방대해지면 한 번 쓰고 버릴 임시 리스트의 크기도 커진다는 문제가 생김. 생성자를 사용하면 데이터를 순환가능하게 변형하므로 메모리 측면에서 훨씬 유리함.

min(), max()와 같은 함수는 key라는 여러 상황에 유용한 인자를 받음(좋은 건가봄)
# 원본: 20을 반환
min_shares = min(s['shares'] for s in portfolio)
# 대안 : {'name': 'AOL', 'shares' : 20 }
min_shares = min(portfolio, key=lamdba s: s[‘shares'])

1.20 여러 매핑을 단일 매핑으로 합치기

[문제]딕셔너리나 매핑이 여러 개 있고 자료 검색이나 데이터 확인을 위해 하나의 합치고 싶음
[해결] 두 딕셔너리가 있는데 a에서 데이터를 검색하고 그 후 b에 데이터가 있는지 검색 —> collections모듈의 ChainMap 클래스 사용
- 중복 키가 있으면 첫번째 매핑의 값 사용
a = {'x':1, 'z':3}
b = {'y':2, ‘z':4}

```
from collections import ChainMap
c = ChainMap(a,b) #a에 있는거 먼저 검색하고 b검색
print(c) #ChainMap({'x': 1, 'z': 3}, {'y': 2, 'z': 4})
print( c['x']) #>>>1
print( c['y']) #>>>2
print( c['z']) #>>>3
```

[토론]
ChainMap은 매핑을 여러개 받아서 하나처럼 보이게 만들어줌. 그렇게 보이는 것뿐 합쳐주는 건 아님
(단지 매핑에 대한 리스트를 유지하면서 리스트를 스캔하도록 일반적인 딕셔너리 동작을 재정의함)

```
len(c) #>>>3
list(c.keys())  #>>>['y', 'x', 'z'] 뭐징 순서는 걍 나오는 건가?
list(c.values()) #>>> [2, 1, 3]
```

매핑의 값을 변경하는 동작은 언제나 리스트의 첫 번째 매핑에 영향을 줌

```
c['z'] = 10
c['w'] = 40
del c['x'] #{'z': 10, 'w': 40} #걍  ChainMap( a딕셔너리, b딕셔너리) 앞에있는 딕셔너리 값을 차례대로 바꿔주는듯
print(a)
del c['y'] #오류남
```

ChainMap은 프로그래밍 언어의 변수와 같이 범위가 있는 값(즉, 전역변수(global), 지역변수)에서 사용하면 유용

```
values = ChainMap()
values['x'] = 1

#새로운 매핑 추가
values = values.new_child()
values['x'] = 2

#새로운 매핑 추가
values = values.new_child()
values['x'] = 3

print(values) # ChainMap({'x': 3}, {'x': 2}, {'x': 1})
print( values['x'] ) # >>> 3

#마지막 매핑 삭제 ( 마지막으로 추가한 매핑이라는 뜻인듯, 추가하면 맨 앞쪽에 위치하게 되나봄)
values = values.parents
values['x'] #>>> 2

#마지막 매핑 삭제
values = values.parents
values['x'] #>>>3

print(values) #>>>ChainMap({'x': 1})
```

ChainMap의 대안으로 update()를 사용해 딕셔너리를 하나로 합칠 수도 있음

```
a = {'x':1, 'z':3}
b = {'y':2, 'z':4}
merged = dict(b) #이건 왜 dict를 만들어주는거야? 별개의 딕셔너리 객체를 새로 만들어야 한다는게 뭔말
merged.update(a) #업데이트에 들어가는 애 기준이구만
print(merged) #{'y': 2, 'z': 3, 'x': 1}
merged['x'] #>>>1
merged['y'] #>>>2
merged['z'] #>>>3
```

—> 잘 작동은 하지만 완전히 별개의 딕셔너리 객체를 새로 만들어야 한다(or 기존 딕셔너리 내용을 변경해야 함) & 원본 딕셔너리의 내용이 변경된다 해도 하쳐놓은 딕셔너리에는 반영되지 않음

```
a['x'] =13
merged['x'] #>>>1
```

```
a = {'x':1, 'z':3}
b = {'y':2, 'z':4}
merged = ChainMap(a,b)
merged['x'] #>>>1
a['x'] = 42
merged['x'] #>>>42
```

Chap2. 문자열과 텍스트

- 문자열 나누기, 검색, 빼기, 렉싱, 파싱과 같은 텍스트 처리
- 대부분 내장 문자열 메소드 사용 . 복잡할 경우 정규식, 문자열 파서 사용
- 유니코드 다루기

2.1 여러 구분자로 문자열 나누기

[문제] 문자열을 필드로 나누고 싶지만 구분자(그리고 그 주변의 공백)가 문자열에 일관적이지 않음
[해결] split() 메소드는 아주 간단한 상황에 사용하도록 설계 되어있음. 여러 개의 구분자나 구분자 주변의 공백까지는 고려하지 않음
—> re.split() 메소드 사용

```
line = 'asdf fjdk; afed, fjek,asdf,     foo'
import re
re.split(r'[;,\s]\s*', line) #>>>>['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
```

[토론]
- re.split()함수는 분리 구문마다 여러 패턴을 명시할 수 있음(ex)쉼표, 세미콜론, 공백문자, 하나 이상의 공백을 분리 문자로 사용)
- 괄호 안에 묶인 정규 표현식 패턴이 캡쳐 그룹이 된다는 점에 주의 해야함. 캡쳐 그룹을 사용하면 매칭된 텍스트에도 결과가 포함됨

```
fields = re.split(r'(;|,|\s)\s*',line)
fields #>>>['asdf', ' ', 'fjdk', ';', 'afed', ',', 'fjek', ',', 'asdf', ',', 'foo']
```

구분 문자만 추출해야 할 필요도 있음.(ex) 출력문 재구성 할 때)

```
values = fields[::2] #이게 뭐여
delimiters = fields[1::2] + ['']
print(values) #>>> ['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
print(delimiters) #>>>[' ', ':', ',', ',', ',', '']

#동일한 구분자로 라인을 구성
print ( ''.join(v+d for v,d in zip(values, delimiters)) ) #>>>asdf fjdk;afed,fjek,asdf,foo
```

분리 구문을 결과에 포함시키고 싶지 않지만 정규 표현식에 사용해야할 필요가 있다면 논캡쳐그룹을 사용해야 한다. (?:…)와 같이 사용하며 다음 코드 참고

```
re.split(r'(?:,|;|\s)\s*',line) #>>>['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo'] #앞에서 한거랑 무슨 차이?
```

2.2 문자열 처음이나 마지막에 텍스트 매칭

[문제] 문자열의 처음이나 마지막에 파일 확장자, URL 스킴 등 특정 텍스트 패턴이 포함되어 있는지 검사하고 싶음
[해결] str.startswich()/ str.endswith()
- 여러 선택지인 경우 튜플에 담아야 함

```
filename = 'spam.txt'
filename.endswith('.txt') #>>>True
filename.startwith('file:') #>>>False
url = 'http://www.python.org'
url.startswith('http://') #>>>True
```

여러 개의 선택지를 검색해야 하면 검사하고 싶은 값을 튜플에 담아 startswith() endswith()에 전달함

```
import os
filenames = os.listdir('.')
print( filenames) #>>>['.DS_Store', '.git', '.idea', 'i_wanna', 'pingpong', 'PythonVariable',
'study_materials', 'tictactoe', 'webcrawling']
[name for name in filenames if name.endswith( ('.c', '.h'))] #업쒀
any(name.endswith('.py') for name in filenames) #>>> True / False 반환
```

```
from urllib.request import urlopen

def read_data(name):
    if name.startswith( ('http://','http:','ftp:')):
        return urlopen(name).read()
    else:
        with open(name) as f:
            return f.read()

choices = ['http://','ftp:']
url = 'http://www.python.org'
url.startswith(choices) # 오류남 --> 튜플로 먼저 변환해줘야함
usl.startswith(tuple(choices)) #True
```

[토론]
startswith() / endswith()메소드는 접두어와 접미어를 검사할 때 매우 편리
슬라이스(slice)를 사용하면 비슷한 동작을 할 수 있지만 가독성이 많이 떨어짐

```
filename = 'spam.txt'
filename[-4:] =='.txt' #>>>True #접미어 : -글자수: #접두어 글자수:
url = 'http://www.python.org'

url[:5] == 'http:' or url[:6] == 'https:' or url[:4] == ' ftp' #>>>True
```

정규식 표현식을 사용해도 됨

```
import re
url = 'http://www.python.org'
print( re.match('http:|https:|ftp:', url) )  #책이랑 다르게 나옴 <_sre.SRE_Match object; span=(0,
5),match='http:'>
```

잘 동작하지만 간단한 매칭을 하기 위해서 복잡한 수식을 작성하는 경우가 많음
—> 레시피에 나온 게 간단하고 빠룸!!!

==> startswitch() / endswith() 메소드는 일반적인 데이터 감소와 같은 다른 동작에 함께 사용하기도 좋음

디렉토리에서 특정 파일이 있는 지를 확인
if any(name.endswith('.c','.h') for name in listdir(dirname)):

2.4 텍스트 패턴 매칭과 검색

[문제] 특정 패턴에 대한 텍스트 매칭이나 검색 하고 싶음
[해결] 매칭하려는 텍스트가 간단하면 : str.find()/ str.startswith(), str.endswith()

```
text = 'yeah, but, no, but no, but yeah'
#정확한 매칭
text == 'yeah'  #>>>False
#처음이나 끝에 매칭
text.startswith('yeah') #>>True
text.endswith('no') #>>False

#처음 나타난 곳 검색
text.find('no') #10
```

복잡한 경우 : 정규 표현식 , re 모듈 사용
(정규 표현식의 기본적인 동작 매커니즘을 이해하기 위해 '11/27/2012' 형식의 날짜가 있다고 가정

```
text1 = '11/27/2012'
text2 = 'Nov 27, 2012'
import re
#간단한 매칭: \d+는 하나 이상의 숫자를 의미
if re.match(r'\d+/\d+/\d+', text1):
    print('yes')
else:
    print('no')

#yes

if re.match(r'\d+/\d+/\d+', text2):
    print('yes')
else:
    print('no')
#no
```

동일한 패턴으로 매칭을 많이 수행할 예정이라면 정규 표현식을 미리 컴파일해서 패턴 객체로 만들어 놓는 것이 좋음

```
datepat = re.compile(r'\d+/\d+/\d+')

if datepat.match(text1):
    print('yes')
else:
    print('no')

if datepat.match(text2):
    print('yes')
else:
    print('no')
```

match()는 항상 문자열 처음에서 찾기를 시도함. 텍스트 전체에 걸쳐 패턴을 찾으려면 **findall()** 메소드 사용

```
text = 'Today is 11/27/2012. PyCon starts 3/13/2013'
datepat.findall(text) #['11/27/2012', '3/13/2013']
```

정규식을 정의할 때 괄호를 사용해 캡쳐 그룹을 만드는 것이 일반적

```
datepat = re.compile(r'(\d+)/(\d+)/(\d+)')
m = datepat.match('11/27/2012')
print(m) #<_sre.SRE_Match object; span=(0, 10), match='11/27/2012'>
```

#각 그룹에서 내용 추출

```
m.group(0) #11/27/2012
m.group(1) #11
m.group(2) #27
m.group(3) #2012
m.groups() #('11','27','2012')
month, day, year = m.groups()
```

#전체 매칭 찾기(튜플로 나눈다)

```
text
datepat.findall(text) #[('11', '27', '2012'), ('3', '13', '2013')] #어디서 튜플로 나눠주는겨??

for month, day, year in datepat.findall(text):
    print('{}-{}-{}'.format(year, month, day))
#2012-11-27
#2013-3-27
```

findall() 메소드는 텍스트를 검색하고 모든 매칭을 찾아 리스트로 반환함.

한번에 결과를 얻지 않고 순환하며 찾으려면 findinter() 사용 ?? 뭔말

```
for m in datepat.findinter(text): #findinter없다고 나옴
    print(m.groups())
#>>>('11','27','2012')
('3','13','2013')
```

[토론]

- re.compile()을 사용해 패턴을 컴파일—> match(), findall(), findinter() 사용
- 패턴을 명시할 때 r'(\d+)/(\d+)/(\d+)'와 같이 raw 문자열을 그대로 쓰는 것이 일반적 (?)
  백슬러시 문자를 해석하지 않고 남겨 두기 때문에 정규 표현식에 유용
  raw 문자열을 사용하지 않으면 r'(\\d+)/(\\d+)/(\\d+)'와 같이 백슬러시를 두번 사용해야 하는 불편함이 따름

- match() 메소드는 문자열의 처음만 확인
m = datepat.match('11/27/2012abcdef')
print(m) #<_sre.SRE_Match object; span=(0, 10), match='11/27/2012'>
print(m.group()) #11/27/2012

datepat = re.compile(r'(\d+)/(\d+)/(\d+)$')
datepat.match('11/27/2012abcdef')
datepat.match('11/27/2012') #1도 모르겠음

- 간단한 텍스트 매칭/검색을 수행하려면 컴파일 과정을 생략하고 re모듈의 모듈레벨 함수를 바로 사용해도 괜춘쓰
re.findall(r'(\d+)/(\d+)/(\d+)', text) #[('11', '27', '2012'), ('3', '13', '2013')]

2.5 텍스트 검색과 치환

[문제] 문자열에서 텍스트 패턴을 검색하고 치환하고 싶음
[해결] 간단한 패턴이라면 str.replace() 메소드 사용

text = "yeah, but, no, but no, but yeah"
text.replace('yeah','yep')
text #>>>'yeah, but, no, but no, but yeah'

복잡한 패턴 사용 시 re모듈의 sub() 함수/메소드 사용
ex)11/27/2012 -> 2012-11-27로 바꿔야할 상황

```
text = 'Today is 11/27/2012. PyCon starts 3/13/2013'
import re
re.sub(r'(\d+)/(\d+)/(\d+)', r'\3-\1-\2', text) #>>>'Today is 2012-11-27. PyCon starts 2013-3-13'
```

\3은 패턴의 캡쳐 그룹 참조

동일한 패턴을 사용한 치환을 계속해야 한다면 성능 향상을 위해 컴파일링을 고려해보는 것이 좋음

```
import re
datepat = re.compile(r'(\d+)/(\d+)/(\d+)')
datepat.sub(r'\3-\1-\2',text) #>>>'Today is 2012-11-27. PyCon starts 2013-3-13' #흠 그렇구나하고 우선 넘어가자
```

더 복잡한 치환을 위해 콜백 함수를 명시할 수도 있음

```
from calendar import month_abbr
def change_date(m):
    mon_name = month_abbr[int(m.group(1))]
    return '{} {} {}'.format(m.group(2), mon_name, m.group(3)) #re.compile 순서대로  m.group ?

datepat.sub(change_date, text) #'Today is 27 Nov 2012. PyCon starts 13 Mar 2013'
```

인자가 되는 치환 콜백은 match()나 find()에서 반환한 매치 객체 사용. 매치에서 특정 부분을 추출하려면 .group() 함수 사용. 치환된 텍스트를 반환해야 함
만약 치환된 텍스트를 받기 전에 치환이 몇 번 발생했는지 알고 싶다면 re.subn()사용

```
newtext, n = datepat.subn(r'\3-\2-\1', text)
newtext #'Today is 2012-27-11. PyCon starts 2013-13-3'
n # 2
```

[토론]
sub() 메소드에 정규 표현식 검색과 치환이 젤 어려움. 정규식 패턴은 따로 공부하셈

2.6 대소문자를 구별하지 않는 검색과 치환

[문제] 텍스트를 검색하고 치환할 때 대소문자를 구별x
[해결] re모듈 사용. re.IGNORECASE 플래그 지정

```
text = 'UPPER PYTHON, lower python, Mixed Python'
re.findall('python', text, flags = re.IGNORECASE) #['PYTHON', 'python', 'Python']
re.sub('python','snake',text,flags =re.IGNORECASE) #'UPPER snake, lower snake, Mixed snake'
```

치환된 텍스트의 대소문자가 원본의 대소문자와 일치하지 않음 -> 함수를 만들어 제공

```
def matchcase(word):
    def replace(m):
        text = m.group()
        if text.isupper():
            return word.upper()
        elif text.islower():
            return word.lower()
        elif text[0].isupper(): #첫글자만 대문자 나머지는 소문자
            return word.capitalize()
        else:
            return word
    return replace ??오잉 이게 무야 replace()?가 아니구 그냥 replace?이게 뭐여

re.sub('python',matchcase('snake'), text, flags=re.IGNORECASE)
#'UPPER SNAKE, lower snake, Mixed Snake'
```

re.IGNORECASE를 사용하는 것만으로 대소문자를 무시한 텍스트 작업에 무리가 없음
유니코드(Unicode)가 포함된 작업을 하기에는 부족함 (2.10)

2.7 가장 짧은 매칭을 위한 정규 표현식

[문제] 정규 표현식을 사용한 텍스트 매칭을 하고 싶지만 텍스트에서 가장 긴 부분을 찾아냄. 가장 짧은 부분을 찾아내고 싶다면 어떠카냐~
[해결] 문자 구분자에 둘러싸여 잇는 텍스트를 찾을 때 종종 발생(인용문같은 경우)

```
str_pat = re.compile(r'\"(.*)\"')
text1 = 'Computer says "no."'
str_pat.findall(text1) #['no.']
text2 = 'Computer says "no." Phone says "yes."'
str_pat.findall(text2) #['no." Phone says "yes.']
```

r'\"(.*)\"' 패턴은 따옴표에 둘러쌓인 텍스트를 찾도록 의도한 것
*문자는 정규 표현식에서 기본적으로 탐욕스럽게 소비되므로 가장 긴 텍스트를 찾게 됨. 그래서 text2에서 원치 않게 인용문 두개에 동시에 매칭함
—> 해결하려면 * 뒤에 ?를 붙이면 됨

```
str_pat = re.compile(r'\"(.*?)\"')
text2 = 'Computer says "no." Phone says "yes."'
str_pat.findall(text2) #['no.', 'yes.']
```

[토론]
.을 사용한 정규 표현식을 작성할 때 가장 일반적인 문제를 보여줌. 패턴에서 점은 개행문(newline)을 제외한 모든 문자를 매칭함.

하지만 점을 텍스트의 시작, 중간 사이에 넣으면 매칭은 가장 긴 것을 찾아내려고 함. 결국 문장의 시작과 끝이 매칭에서 무시되고 더 긴 매칭에 하나로 포함되어 버림. 이때 *나 +에 ?를 붙이면 매칭 알고리즘에게 가장 짧은 것을 찾아내도록 명시할 수 있음

2.8 여러 줄에 걸친 정규 표현식 사용

[문제] 여러 줄에 걸친 정규 표현식 매칭을 사용하고 싶음
[해결] 점(.)을 사용한 텍스트 매칭을 할 때 이 문자가 개행문에 매칭하지 않는 다는 사실을 잊었을 때 일반적으로 발생.
예를 들어 다음과 같이 텍스트에서 C스타일 주석을 찾아보자

```
comment = re.compile(r'/\*(.*?)\*/')
text1 = '/* this is a comment */'
text2 = '''/*this is a
            multiline comment */
'''
comment.findall(text1)  #[' this is a comment ']
comment.findall(text2)  #[]
```

text2에 C스타일 주석이 포함되어 있지만 못 찾아냄. 개행문을 패턴에 넣어야 함

```
comment = re.compile(r'/\*((?:.|\n)*?)\*/')
comment.findall(text2) #['this is a\n                multiline comment ']
```

(?:.|\n)은 논캡쳐 그룹을 명시(매칭의 명식은 명시하지만 개별적으로 캡쳐하거나 숫자를 붙이지 않음)
[토론]
re.compile()함수에 re.DOTALL이라는 유용한 플래그를 사용할 수 있음 .이 플래그를 사용하면 정규 표현식의 점(.) 이 개행문을 포함한 모든 문자에 매칭함

```
comment = re.compile(r'/\*((?:.|\n)*?)\*/',re.DOTALL)
comment.findall(text2) #['this is a\n                multiline comment ‘] #위랑 뭐가 다름?
```

re.DOTALL 플래그를 사용하면 간단한 패턴에는 잘 작동. 복잡한 패턴 사용& 여러 정규식 합쳐 토큰화 할 때 문제가 생길 수 있음(2.18참고)
=> 플래그 없이 잘 동작할 수 있도록 정규 표현식을 작성하는 것이 더 좋음

2.11 문자열에서 문자 잘라내기

[문제] 텍스트의 처음, 끝, 중간에서 원하지 않는 공백문을 잘라내고 싶음
[해결] strip() 메소드를 사용하면 가능. lstrip(), rstrip(), 공백을 잘라내지만 원하는 문자를 지정할 수도 있음

```
s = '     hello world \n'
s.strip() #'hello world’  #공백 있으면 다 짤라내네
s.lstrip() #'hello world \n'
s.rstrip() #'     hello world'

t = '-----hello====='
t.lstrip('-') #’hello====' #왼쪽 -이면 잘라냄  ‘---0--hello====' 이면 —-만 없앰
t.strip('-=') #'hello'
```

[토론]
데이터를 보기 좋게 만들기 위한 용도로 여러 strip()메소드를 일반적으로 사용함. 예를 들어 문자열에서 공백문을 없애거나 인용 부호를 삭제하는 식 but, 텍스트 중간에서 잘라내기를 할 수는 없음

```
s = '    hello    world   \n'
s = s.strip()
s #'hello    world'

문자열 중간의 공백문이 사라지지 않았음 —-> replace()메소드나 정규 표현식의 치환 사용

s.replace(' ','') #'helloworld'
import re
re.sub('\s+', ' ',s) #'hello world'
```

때로는 파일을 순환하며 데이터를 읽어 들이는 것과 같이 다른 작업과 문자열을 잘라내는 작업을 동시에 하고 싶을 수 있음 —> 생성자 표현식 사용

```
with open(filename) as f:
    lines = (line.strip() for line in f) #데이터 변환 담당
    for line in lines:
...
```

-> 실질적인 임시 리스트를 만들지 않으므로 효율적. 단지 잘라내기 작업이 적용된 라인은 순환하는 이터레이터를 생성할뿐 (고급 기술로는 translate()가 있음)

2.12 텍스트 정리

[문제] 누가 텍스트 이상하게 바꿔놓음 ”pýt̂hön”
[해결] 텍스트 정리 작업은 대개 텍스트 파싱과 데이터 처리와 관련이 있음
단순히 생각하면 기본적인 문자열 함수(str.upper() , str.lower())을 사용해서 텍스트 표준 케이스로 변환하면 됨
str.replace()나 .re.sub()을 사용한 치환은 특정 문자 시퀀스를 아예 없애거나 바꾸는 데만 집중할 수 있다.
—>고급진 방법이 있음. str.tarnslate()

```
s = 'pýt̂hön\fis\tawesome\r\n'
s #'pýt̂hön\x0cis\tawesome\r\n'

# 1) 공백문 제거 : 작은 변환 테이블을 만들어 놓고 translate() 사용
remap = {
    ord('\t') : ' ',
    ord('\f') : ' ',
    ord('\r') : None #삭제됨
}
a = s.translate(remap)
a #'pýt̂hön is awesome\n'
```

발전시켜서 더 큰 변환 테이블을 만들 수 있음. 예를 들어 결합 문자를 모두 없앨 수 있음 엉엉…몰라…

```
import unicodedata
import sys
cmb_chrs = dict.fromkeys(c for c in range(sys.maxunicode) if unicodedata.combining(chr(c)))
#dict.fromkeys()를 사용해 딕셔너리가 모든 유니코드 결합 문자를 None으로 매핑하고 있음

b = unicodedata.normalize('NFD',a)
b #'pýt̂hön is awesome\n'
b.translate(cmb_chrs) #'python is awesome\n' #변환 함수를 사용해 필요없는 함수 모두 삭제
```

컨트롤 문자와 같이 다른 문자를 없앨 때 응용이 가능

유니코드 숫자 문자를 이와 관련 있는 아스크 숫자에 매핑하도록 변환 테이블을 작성 몰라

```
digitmap = { c: ord('0') + unicodedata.digit(chr(c)) for c in range(sys.maxunicode) if
unicodedata.category(chr(c)) == 'Nd' }
len(digitmap) #460
# 아라비아 숫자
x = '\u0661\u0662\u0663'
x.translate(digitmap) #>>>'123'
```

```
a #'pýt̂ön̈ is awesome\n'
b = unicodedata.normalize('NFD', a)
b.encode('ascii', 'ignore').decode('ascii') #'python is awesome\n'
```

[토론]
텍스트를 정리하다보면 실행 성능 문제에 직면함. 간단한 치환을 하려면 str.replace메소드 사용
공백을 없애려면 다음과 같이 함

```
def clean_spaces(s):
    s = s.replace('\r','')
    s = s.replace('\t',' ')
    s = s.replace('\f',' ')
    return s
```

translate() ,정규보다 빠름
translate()는 복잡한 리매핑이나 삭제에 사용하면 빠름. 알아서 잘 쓰셈

2.13 텍스트 정렬

[문제] 텍스트 특정 형식에 맞추어 정렬
[해결] ljust(), rjust(), center()

```
text = 'Hello World'
text.ljust(20)
text.rjust(20)
text.center(20)

text.ljust(20,'=')
text.center(20,'*')

format(text,'>20')
format(text,'<20')
format(text,'^20')

format(text,'=>20')
format(text,'*^20')

'{:>10s} {:>10s}'.format('Hello','World') #'     Hello      World'

x = 1.2345
format(x, '>10') #'    1.2345'
format(x,'^10.2f') #'   1.23   '
```

forma()은 단순히 ljust(), rjust(), center()메소드에 비해 더 일반적인 목적에 사용할 수 있고 모든 객체에 동작함

2.14 문자열 합치기

[문제] 작은 문자열 여러개 합쳐서 긴 문자열 생성
[해결] join()

```
parts = ['Is','Chicago', 'Not', 'Chicago?']
' '.join(parts) #'Is Chicago Not Chicago?'
```

```
a = 'Is Chicago'
b = 'Not Chicago'
print('{} {}'.format(a,b))

#소스 코드에서 문자열을 합치려고 할 때는 단순히 옆에 붙여 놓여 놓기만 해도 됨
a = 'Hello' 'World'
a #HelloWorld
```

문자열 코드 합치기 작성하지 마셈

```
s = ''
for p in parts:
    s += p

#생성자 표현식
data = ['ACME', 50, 91.1]
','.join(str(d) for d in data) #'ACME,50,91.1'

print( a,b,c,sep=':')

#모르겠음….그냥 나중에 하자…..
```

2.15 문자열에 변수 사용

[문제] 문자열에 변수를 사용하고 이 변수에 맞는 값을 채우고 싶음
[해결] 파이썬 문자열에 변수 값을 치환하는 간단한 방법은 없음, format()메소드 사용

```
s = '{name} has {n} messages.'
s.format(name='Guido', n=37)
```

치환할 값이 변수에 들어있다면 format_map()과 vars()를 함께 사용하면 됨

```
name = 'Guido'
n = 37
s.format_map( vars() ) #이게 어떻게 되는 건지 모르겠음
```

var()에는 인스턴스를 사용할 수도 있음

```
class Info:
    def __init__(self,name,n):
        self.name = name
        self.n = n

a = Info('Guido',37)
s.format_map(vars(a)) #'Guido has 37 messages.
```

format()과 format_map()을 사용할 때 빠진 값이 있으면 제대로 동작하지 않음 , __missing__() 메소드가 있는 딕셔너리 클래스를 정의해서 피할 수 있음
s.format(name='Guido') #KeyError: 'n'

```
class safesub(dict):
    def __missing__(self,key):
        return '{' + key + '}'
del n # n이 정의되지 않도록 함 뭐여
s.format_map(safesub(vars())) #'Guido has {n } messages.' #vars() 얘가 어떻게 되는 지 모르겠어 key?
```

코드에서 변수 치환을 빈번히 사용할 것 같으면 치환하는 작업을 유틸리티 함수에 모아놓고 프레임 핵으로 사용할 수 있음

```
import sys
def sub(text):
    return text.format_map(safesub(sys._getframe(1).f_locals))
            #sys._getframe(1) : 호출자의 스택 프레임 반환
            #지역변수를 얻기 위해 f-locals요소에 접근함
            #( 대개의 코드에서 스택 프레임에 접근하는 것을 권장 하지 않음)  ???
            # but, 문자열 치환 기능과 같은 유틸리티 함수에서는 유용할 수 도 있음
            #f_locals : 호출함수의 지역변수 복사본을 담아 둔 딕셔너리

name = 'Guido'
n =37
print( sub('Hello {name}')) #Hello Guido
print(sub('You have {n} messages.')) #You have 37 messages.
print(sub('Your favorite color is {color}')) #Your favorite color is {color }
```

[토론]
- 파이썬 자체에서 변수 보간법이 존재하지 않아서 방법이 많음
%문자열 서식화 , template 사용법도 있지만 format(0, format_map() 메소드 사용을 더 권장
—>format(0 사용시 문자열 서식화와 관련 있는 모든 기능(절렬, 공백, 숫자 서식) (template 문자열 객체에서는 지원하지 않음)

- 매핑, 딕셔너리의 거의 알려지지 않은 __missing__() 메소드를 통해 없는 값을 처리할 수 있음
        safesub클래스에서 이 메소드를 정의하여 없는 값을 기본으로 처리하도록 했음.
        —> KeyError 예외가 발생하지 않고 값이 없음을 알리는 문자열을 반환(디버깅할때 유용함)

2.16 텍스트 열의 개수 고정

[문제] 긴 문자열의 서식을 바꿔 열의 개수 조정
[해결] textwrap 모듈을 사용해서 reformat
- 터미널에 사용할 텍스트에 적합

```
[터미널 크기 얻기]
import os
os.get_terminal_size().columns
#내컴에서 오류 Traceback (most recent call last):
#File "<input>", line 2, in <module>
```

- fill() 메소드 사용시 탭처리, 문장의 끝과 같은 추가관리 가능 text wrap.TextWrapper 참고

```
s = "Look into my eyes, look into my eyes, the eyes, the eyes, \
the eyes, not around the eyes, don't look around the eyes, \
look into my eyes, you're under."
import textwrap
print(textwrap.fill(s, 70))
# Look into my eyes, look into my eyes, the eyes, the eyes, the eyes,
# not around the eyes, don't look around the eyes, look into my eyes,
# you're under.

print(textwrap.fill(s, 40))
# Look into my eyes, look into my eyes,
# the eyes, the eyes, the eyes, not around
# the eyes, don't look around the eyes,
# look into my eyes, you're under.

print(textwrap.fill(s, 40, initial_indent='    '))
#     Look into my eyes, look into my
# eyes, the eyes, the eyes, the eyes, not
# around the eyes, don't look around the
# eyes, look into my eyes, you're under.

print(textwrap.fill(s, 40, subsequent_indent='    '))
# Look into my eyes, look into my eyes,
#     the eyes, the eyes, the eyes, not
#     around the eyes, don't look around
#     the eyes, look into my eyes, you're
#     under.
```