벡터화된 문자열 연산

파이썬의 강점 중 하나는 문자열 데이터를 처리하고 가공하기가 비교적 쉽다는 것이다. Pandas 는 파이썬을 기반으로 만들어져 현실 세계의 데이터로 작업할 때(작업이라 쓰고 정제라 읽는다) 필요한 먼징 유형의 핵심 부분인 벡터화된 문자열 연산을 종합적으로 제공한다.
몇가지 Pandas 문자열 연산을 검토하고 나서 그것들을 활용해 인터넷에서 수집한 아주 지저분한 조리법 데이터세트를 부분적으로 정제하는 과정을 보자. .

Pandas 문자열 연산소개

NumPy 와 Pandas 같은 도구가 산술 연산을 어떻게 일반화해서 수 많은 배열 요소에 동일한 연산을 쉽고 빠르게 수행하는지 살펴보자

```
In [1]:import numpy as np
       x = np.array([2, 3, 5, 7, 11, 13])
       x * 2
Out[1]:array([ 4,  6, 10, 14, 22, 26])
```

이렇게 연산을 벡터화하면 데이터배열에서 동작하는 구문이 단순해진다. 더는 배열의 크기나 모양을 걱정할 필요없이 하고자 하는 연산이 무엇인지만 신경쓰면 된다. NumPy 에서는 문자열배열에 그렇게 간단히 접근할 수 없으므로 계속해서 장황한 루프 구문을 사용해야 한다.

```
In [2]:data = ['peter', 'Paul', 'MARY', 'gUIDO']
       [s.capitalize() for s in data]
Out[2]:['Peter', 'Paul', 'Mary', 'Guido']
```

일부 데이터에서는 이것만으로도 충분하겠지만 누락된 값이 있다면 에러가 발생할 것이다.

```
In [3]:data = ['peter', 'Paul', None, 'MARY', 'gUIDO']
       [s.capitalize() for s in data]
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-3-fc1d891ab539> in <module>()
     1 data = ['peter', 'Paul', None, 'MARY', 'gUIDO']
----> 2 [s.capitalize() for s in data]

<ipython-input-3-fc1d891ab539> in <listcomp>(.0)
     1 data = ['peter', 'Paul', None, 'MARY', 'gUIDO']
----> 2 [s.capitalize() for s in data]

AttributeError: 'NoneType' object has no attribute 'capitalize'
```

Pandas 는 문자열을 담고 있는 Pandas Series 와 Index 객체의 str 속성을 통해 벡터화된 문자열 연산을 수행하고 누락된 데이터를 올바르게 처리하기 위한 기능을 제공한다. 그렇다면 예를 들어 이 데이터로 Pandas Series 를 생성했다고 가정하자.

```
In [4]:import pandas as pd
       names = pd.Series(data)
       names
Out[4]:0    peter
       1     Paul
       2     None
       3     MARY
       4    gUIDO
       dtype: object
```

이제 누락된 값은 건너뛰면서 모든 항목의 첫글자를 대문자로 변경하는 메서드를 호출할 수 있다.

```
In [5]:names.str.capitalize()
Out[5]:0    Peter
       1     Paul
       2     None
       3     Mary
       4    Guido
       dtype: object
```

이 str 속성에 탭 자동완성을 사용하면 Pandas 에 사용할 수 있는 모든 벡터화된 문자열 메서드의 목록을 볼 수 있다.

Pandas 문자열 메서드 목록

파이썬에서 문자열 조작 방법을 잘 알고 있다면 Pandas 의 문자열 구문 대부분은 직관적이어서 사용할 수 있는 메서드 목록만 제공해도 충분할 것이다.

```
In [6]:monte = pd.Series(['Graham Chapman', 'John Cleese', 'Terry Gilliam',
                          'Eric Idle', 'Terry Jones', 'Michael Palin'])
```

파이썬 문자열 메서드와 유사한 메서드

거의 모든 파이썬의 내장 문자열 메서드는 Pandas 의 벡터화된 문자열 메서드에도 반영돼 있다. 다음은 파이썬의 문자열 메서드를 반영한 Pandas str 메서드의 목록이다.

len()	lower()	translate()	islower()

ljust()	upper()	startswith()	isupper()
rjust()	find()	endswith()	isnumeric()
center()	rfind()	isalnum()	isdecimal()
zfill()	index()	isalpha()	split()
strip()	rindex()	isdigit()	rsplit()
rstrip()	capitalize()	isspace()	partition()
lstrip()	swapcase()	istitle()	rpartition()

이 메서드들은 다양한 변화 값을 가진다는 점을 알아두자. lower()같은 일부 메서드는 일련의 문자열을 반환한다.:

```
In [7]:monte.str.lower()
Out[7]:0    graham chapman
       1       john cleese
       2    terry gilliam
       3        eric idle
       4      terry jones
       5    michael palin
       dtype: object
```

그러나 일부 다른 메서드는 숫자를 반환한다.

```
In [8]:monte.str.len()
Out[8]:0    14
       1    11
       2    13
       3     9
       4    11
       5    13
       dtype: int64
```

또는 부울로 반환한다.

```
In [9]:monte.str.startswith('T')
```

```
Out[9]:0    False
       1    False
       2     True
       3    False
       4     True
       5    False
       dtype: bool
```

또 다른 메서드는 각 요소에 대한 리스트나 다른 복합 값을 반환한다.

```
In [10]:monte.str.split()
Out[10]:0    [Graham, Chapman]
       1      [John, Cleese]
       2    [Terry, Gilliam]
       3       [Eric, Idle]
       4      [Terry, Jones]
       5    [Michael, Palin]
       dtype: object
```

정규표현식을 활용하는 메서드

그 밖에도 각 문자열 요소의 내용을 검사하는 데 정규 표현식을 활용하고 파이썬에서 기본적으로 제공하는 re 모듈의 API 규칙을 일부 따르는 메서드도 있다.

Method	Description
`match()`	Call `re.match()` on each element, returning a boolean.
`extract()`	Call `re.match()` on each element, returning matched groups as strings.
`findall()`	Call `re.findall()` on each element
`replace()`	Replace occurrences of pattern with some other string
`contains()`	Call `re.search()` on each element, returning a boolean
`count()`	Count occurrences of pattern
`split()`	Equivalent to `str.split()`, but accepts regexps

Method	Description
`rsplit()`	Equivalent to `str.rsplit()`, but accepts regexps

이 메서드를 사용하면 흥미로운 연산을 다양하게 수행할 수 있다. 예들 들어. 각 요소의 시작 문자와 붙어 있는 그들을 요청해 각 요소로부터 이름 부분을 추출할 수 있다.

```
In [11]: monte.str.extract('([A-Za-z]+)', expand=False)
Out[11]: 0     Graham
         1      John
         2     Terry
         3      Eric
         4     Terry
         5   Michael
         dtype: object
```

또는 문자열 시작(^)과 끝($)을 나타내는 정규 표현식을 사용해 처음으로 시작으로 시작하고 끝나는 모든 이름을 찾는 것과 같은 더 복잡한 일을 할 수도 있다.

```
In [12]: monte.str.findall(r'^[^AEIOU].*[^aeiou]$')
Out[12]: 0    [Graham Chapman]
         1                  []
         2    [Terry Gilliam]
         3                  []
         4      [Terry Jones]
         5    [Michael Palin]
         dtype: object
```

`Series`나 `Dataframe`항목에 정규 표현식을 간결하게 적용할 수 있게 되면서 데이터 정제와 분석에 있어 많은 가능성이 열렸다.

기타 메서드

마지막으로 다른 편리한 연산이 가능한 메서드가 몇 가지 있다.

Method	Description
`get()`	Index each element
`slice()`	Slice each element

Method	Description
`slice_replace()`	Replace slice in each element with passed value
`cat()`	Concatenate strings
`repeat()`	Repeat values
`normalize()`	Return Unicode form of string
`pad()`	Add whitespace to left, right, or both sides of strings
`wrap()`	Split long strings into lines with length less than a given width
`join()`	Join strings in each element of the Series with passed separator
`get_dummies()`	extract dummy variables as a dataframe

백터화된 항목의 접근 및 슬라이싱

get()과 slice()연산은 특히 각 배열 각 배열에서 벡터화된 요소에 접근하게 해준다. 예를 들어, str.slice(0,3)을 사용하면 각 배열의 첫 세 문자의 슬라이스를 얻을 수 있다. 이 행위는 파이썬의 일반 인덱싱 구문으로도 수행할 수 있다. 예를 들면, df.str.slice(0,3)은 df.str.[0:3]과 동일하다.

```
In [13]:monte.str[0:3]
Out[13]:0    Gra
        1    Joh
        2    Ter
        3    Eri
        4    Ter
        5    Mic
        dtype: object
```

`df.str.get(i)` 와 `df.str[i]` 를 통한 인덱싱은 유사하다.,
이 get()과 slice()메서드를 이용하면 split()이 반환한 배열의 요소에 접근할 수도 있다. 예를 들어, 각 요소의 성을 추출하려면 slice()과 get()을 결합하면 된다.

```
In [14]:monte.str.split().str.get(-1)
Out[14]:0    Chapman
        1     Cleese
```

```
2    Gilliam
3       Idle
4      Jones
5      Palin
dtype: object
```

지시 변수

별도의 설명이 약간 필요한 또 다른 메서드로 get_dummies()메서드가 있다. 이 메서드는 데이터가 일종의 코딩된 지시자를 포함한 열을 가지고 있을 때 유용하다. 예를 들면 데이터세트가 A="born in America", B="born in the United Kingdom",C="likes chees",D="likes span"과 같이 코드 형태의 정보를 포함할 수도 있다.

```
In [15]: full_monte = pd.DataFrame({'name': monte,
                                    'info': ['B|C|D', 'B|D', 'A|C',
                                             'B|D', 'B|C', 'B|C|D']})
         full_monte
Out[15]:
```

	Info	name
0	B|C|D	Graham Chapman
1	B|D	John Cleese
2	A|C	Terry Gilliam
3	B|D	Eric Idle
4	B|C	Terry Jones
5	B|C|D	Michael Palin

`get_dummies()` 루틴을 이용하면 이 지시 변수를 `DataFrame`으로 신속하게 나눌 수 있다.

```
In [16]: full_monte['info'].str.get_dummies('|')
Out[16]:
```

	A	B	C	D
0	0	1	1	1
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	0	1	1	0
5	0	1	1	1

이 연산은 기본 구성요소로 하면 데이터를 정제할 때 끊임없는 문자열 처리 절차를 구성할 수 있다.

이 메서드에 대해서는 더 이상 깊이 다루지 않겠지만, Pandas 온라인 문서의 텍스트 데이터로 작업하기를 읽어보면 좋다.