정보교육을 위한 파이썬

정보 탐색

Version 0.0.9-d2

저자: Charles Severance
번역: 이광춘, 한정수
(xwmooc)

Printing history:

October 2013: Major revision to Chapters 13 and 14 to switch to JSON and use OAuth. Added new chapter on Visualization.

September 2013: Published book on Amazon CreateSpace

January 2010: Published book using the University of Michigan Espresso Book machine.

December 2009: Major revision to chapters 2-10 from *Think Python: How to Think Like a Computer Scientist* and writing chapters 1 and 11-15 to produce *Python for Informatics: Exploring Information*

June 2008: Major revision, changed title to *Think Python: How to Think Like a Computer Scientist*.

August 2007: Major revision, changed title to *How to Think Like a (Python) Programmer*.

April 2002: First edition of *How to Think Like a Computer Scientist*.

The LaTeX source for the *Think Python: How to Think Like a Computer Scientist* version of this book is available from `http://www.thinkpython.com`.

Chapter 1

튜플(Tuples)

1.1 튜플은 불변하다.

A tuple[1] is a sequence of values much like a list. The values stored in a tuple can be any type, and they are indexed by integers. The important difference is that tuples are **immutable**. Tuples are also **comparable** and **hashable** so we can sort lists of them and use tuples as key values in Python dictionaries.

Syntactically, a tuple is a comma-separated list of values:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Although it is not necessary, it is common to enclose tuples in parentheses to help us quickly identify tuples when we look at Python code:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

To create a tuple with a single element, you have to include the final comma:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

Without the comma Python treats `('a')` as an expression with a string in parentheses that evaluates to a string:

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
```

Another way to construct a tuple is the built-in function `tuple`. With no argument, it creates an empty tuple:

[1] Fun fact: The word "tuple" comes from the names given to sequences of numbers of varying lengths: single, double, triple, quadruple, quituple, sextuple, septuple, etc.

```
>>> t = tuple()
>>> print t
()
```

If the argument is a sequence (string, list or tuple), the result of the call to `tuple` is a tuple with the elements of the sequence:

```
>>> t = tuple('lupins')
>>> print t
('l', 'u', 'p', 'i', 'n', 's')
```

Because `tuple` is the name of a constructor, you should avoid using it as a variable name.

Most list operators also work on tuples. The bracket operator indexes an element:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print t[0]
'a'
```

And the slice operator selects a range of elements.

```
>>> print t[1:3]
('b', 'c')
```

But if you try to modify one of the elements of the tuple, you get an error:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

You can't modify the elements of a tuple, but you can replace one tuple with another:

```
>>> t = ('A',) + t[1:]
>>> print t
('A', 'b', 'c', 'd', 'e')
```

1.2 튜플 비교하기

비교연산자는 튜플과 다른 열(sequence)에도 동작한다. 파이썬은 각 열로부터 첫 요소를 비교하는 것에서부터 비교를 시작한다. 만약 두 요소가 같다면, 다음 요소로 비교를 진행하여 다른 요소를 찾을 때까지 계속한다. 후속 요소는 얼마나 큰 값인지에 관계없이 비교 고려대상은 아니다.

```
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
```

`sort` 함수도 동일한 방식으로 작동한다. 첫 요소를 먼저 정렬하지만, 동일한 경우 두 번째 요소를 정렬하고, 그 후속 요소를 동일한 방식으로 정렬한다. 이 기능은 다음 **DSU**라고 불리는 패턴에 적용된다.

데코레이트(Decorate) 열로부터 요소를 선행하는 하나 혹은 그 이상의 키를 가진 튜플 리스트를 구축하는 열

정렬(Sort) 파이썬 내장 함수 sort를 사용한 튜플 리스트

언데코레이트(Undecorate) 열의 정렬된 요소를 추출.

예를 들어, 단어 리스트가 있고 가장 긴 단어부터 가장 짧은 단어 순으로 정렬한다고 가정하자.

```
txt = 'but soft what light in yonder window breaks'
words = txt.split()
t = list()
for word in words:
   t.append((len(word), word))

t.sort(reverse=True)

res = list()
for length, word in t:
    res.append(word)

print res
```

첫 루프는 튜플 리스트를 생성하고, 각 튜플은 선행하여 길이를 가진 단어다.

sort가 첫 요소, 길이를 우선 비교하고, 동률일 경우 두 번째 요소를 고려한다. sort 함수의 인수 reverse=True는 내림차순으로 정렬한다.

두 번째 루프는 튜플 리스트를 훑고, 내림차순 길이 순으로 리스트를 생성한다. 그래서, 5 문자 단어는 역 알파벳 순으로 정렬되어 있다. 다음 리스트에서 “what”이 “soft” 보다 앞에 나타난다.

프로그램의 출력은 다음과 같다.

```
['yonder', 'window', 'breaks', 'light', 'what',
'soft', 'but', 'in']
```

물론, 파이썬 리스트로 변환하여 내림차순 길이 순으로 정렬된 문장은 시적인 의미를 많이 잃어버렸다.

1.3 튜플 할당

파이썬 언어의 독특한 구문론적인 기능중의 하나는 할당문의 왼편에 튜플을 놓을 수 있는 것이다. 왼쪽 편이 열인 경우 한번에 하나 이상의 변수를 할당할 수 있게 해준다.

다음 예제에서, 열인 두개 요소 리스트가 있고 하나의 명령문으로 변수 x와 y에 열의 첫번째와 두번째 요소를 할당한다.

```
>>> m = [ 'have', 'fun' ]
>>> x, y = m
>>> x
'have'
>>> y
'fun'
>>>
```

마술이 아니다. 파이썬은 *대략* 튜플 할당 구문을 다음과 같이 해석한다.[2]

```
>>> m = [ 'have', 'fun' ]
>>> x = m[0]
>>> y = m[1]
>>> x
'have'
>>> y
'fun'
>>>
```

문체적 할당문의 왼편에 튜플을 사용할 때, 괄호를 생략한다. 하지만 다음은 동일하게 적합한 구문이다.

```
>>> m = [ 'have', 'fun' ]
>>> (x, y) = m
>>> x
'have'
>>> y
'fun'
>>>
```

튜플 할당문을 사용하는 특히 똑똑한 응용사례는 하나의 명령문으로 두 변수의 값을 **교체(swap)**하는 것이다.

```
>>> a, b = b, a
```

양쪽 모두 튜플이지만, 왼편은 튜플의 변수이고 오른편은 튜플의 표현식이다. 오른편의 값이 왼편의 해당하는 변수에 할당된다. 오른편의 모든 표현식은 할당이 이루어지기 이전에 평가된다. 왼편의 변수의 숫자와 오른편의 값의 숫자는 동일해야 한다.

```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

좀더 일반적으로 오른 편은 임의의 열(문자열, 리스트 혹은 튜플)이 될 수 있다. 예를 들어, 전자우편 주소를 사용자 이름과 도메인으로 쪼개기 위해서 다음과 같이 프로그램을 작성할 수 있다.

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
```

`split` 함수로부터 반환값은 두개의 요소를 가진 리스트다. 첫번째 요소는 `uname`에 두번째 요소는 `domain`에 할당된다.

[2] 파이썬은 구문을 문자그대로 해석하지는 않는다. 예를 들어, 동일한 것을 딕셔너리로 작성한다면, 예상한 것처럼 작동하지는 않는다.

```
>>> print uname
monty
>>> print domain
python.org
```

1.4 딕셔너리와 튜플

딕셔너리는 튜플의 리스트를 반환하는 items 메쏘드가 있다. 각 튜플은 키-밸류 페어다.[3]

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = d.items()
>>> print t
[('a', 10), ('c', 22), ('b', 1)]
```

딕셔너리로부터 기대했듯이, 항목은 특별한 순서가 없다.

하지만 튜플 리스트는 리스트여서 비교가 가능하기 때문에, 튜플 리스트를 정렬할 수 있다. 딕셔너리를 튜플 리스트로 변환하는 방법은 키로 정렬된 딕셔너리 내용을 출력하는 것이다.

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = d.items()
>>> t
[('a', 10), ('c', 22), ('b', 1)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

새로운 리스트는 키 값으로 오름차순 알파벳 순으로 정렬된다.

1.5 딕셔너리로 다중 할당

items 함수, 튜플 할당문, for문을 조합해서, 하나의 루프로 딕셔너리의 키와 값을 훑는 멋진 코드 패턴을 만들 수 있다.

```
for key, val in d.items():
    print val, key
```

이 루프는 두개의 **반복 변수(iteration variables)**를 가진다. items 함수는 튜플 리스트를 반환하고, key, val는 딕셔너리의 키-밸류 페어 각각을 성공적으로 반복하는 튜플 할당을 수행한다.

매번 루프를 반복할 때마다, key와 value는 여전히 해쉬 순으로 되어 있는 딕셔너러의 다음 키-밸류 페어로 진행한다.

루프의 출력은 다음과 같다.

[3] 파이썬 3.0으로 가면서 살짝 달라졌다.

```
10 a
22 c
1 b
```

다시한번 해쉬 키 순서다. 즉, 특별한 순서가 없다.

두 기술을 조합하면, 딕셔너리 내용을 키-밸류 페어에 저장된 값의 순서로 정렬하여 출력할 수 있다.

이것을 수행하기 위해서, 각 튜플이 `(value, key)`인 튜플 리스트를 작성한다. `items` 메쏘드를 사용하여 리스트 `(key, value)` 튜플을 만든다. 하지만 이번에는 키가 아닌 값으로 정렬한다. 키-밸류 튜플 리스트를 생성하면, 역순으로 리스트를 정렬하고 새로운 정렬 리스트를 출력하는 것은 쉽다.

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> l = list()
>>> for key, val in d.items() :
...     l.append( (val, key) )
...
>>> l
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> l.sort(reverse=True)
>>> l
[(22, 'c'), (10, 'a'), (1, 'b')]
>>>
```

조심스럽게 각 튜플의 첫 요소로 값을 가지는 튜플 리스트를 생성함으로서 튜플 리스트를 정렬하여 값으로 정렬된 딕셔너리를 얻었다.

1.6 가장 빈도수가 높은 단어

로미오와 쥴리엣 2장 2막 텍스트 파일로 다시 돌아와서, 텍스트에 가장 빈도수가 높은 단어를 10개를 출력하기 위해서 이 기법을 사용하여 프로그램을 보강해보자.

```
import string
fhand = open('romeo-full.txt')
counts = dict()
for line in fhand:
    line = line.translate(None, string.punctuation)
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

# Sort the dictionary by value
lst = list()
for key, val in counts.items():
    lst.append( (val, key) )
```

```
lst.sort(reverse=True)

for key, val in lst[:10] :
    print key, val
```

파일을 일고 각 단어를 문서의 단어 빈도수에 매핑(사상)하는 딕셔너리를 계산하는 프로그램 첫 부분은 바뀌지 않는다. 하지만, counts 를 단순히 출력하는 대신에 (val, key) 튜플 리스트를 생성하고 역순으로 리스트를 정렬한다.

값이 처음이기 때문에, 비교를 위해서 값이 사용되고, 만약 동일한 값을 가진 튜플이 하나이상 존재한다면, 두번째 요소 (키)를 살펴보게 되어서 값이 동일한 경우 키의 알파벳 순으로 추가적으로 정렬이 된다.

마지막에 다중 할당 반복을 수행하는 멋진 for 루프를 작성하고 리스트 쪼개기 (lst[:10])를 통해 가장 빈도수가 높은 상위 10개 단어를 출력한다.

이제 마지막 출력문은 단어 빈도 분석에서 원하는 것을 완수한 것처럼 보인다.

```
61 i
42 and
40 romeo
34 to
34 the
32 thou
32 juliet
30 that
29 my
24 thee
```

복잡한 데이터 파싱과 분석 작업이 이해하기 쉬운 19줄의 파이썬 프로그램으로 수행된 사실이 왜 파이썬이 정보 탐색의 언어로서 좋은 선택인지 보여준다.

1.7 딕셔너리 키로 튜플 사용하기

튜플은 **해쉬가능(hashable)**하고, 리스트는 그렇지 못하기 때문에, 딕셔너리에 사용할 **복합(composite)**키를 생성하려면, 키로 튜플을 사용해야 한다.

만약 성(last-name)과 이름(first-name) 짝을 가지고 전화번호에 매핑(사상)하는 전화번호부를 생성하려고 하면, 복합키를 마주친다. 변수 last, first, number 을 정의했다고 가정하면, 다음과 같이 딕셔너리 할당문을 작성할 수 있다.

```
directory[last,first] = number
```

꺾쇠 괄호의 표현은 튜플이다. 딕셔너리를 훑기 위해서 for 루프에 튜플 할당을 사용한다.

```
for last, first in directory:
    print first, last, directory[last,first]
```

튜플인 directory에 키를 루프가 훑는다. 각 튜플 요소를 last, first에 할당하고, 이름과 해당 전화번호를 출력한다.

1.8 열 : 문자열, 리스트, 튜플

여기서 튜플 리스트에 초점을 맞추었지만, 거의 모든 예제가 또한 리스트의 리스트, 튜플의 튜플, 튜플의 리스트에도 동작한다. 가능한 조합을 열거하는 것을 피하기 위해서 시퀀스의 시퀀스 (sequences of sequences)에 대해서 논의하는 것이 때로는 편리하다.

대부분의 맥락에서 다른 종류의 시퀀스(문자열, 리스트, 튜플)는 상호 호환해서 사용될 수 있다. 그래서 왜 어떻게 다른 것보다 이것을 선택해야 될까요?

명확하게 시작하기 위해서, 문자열은 요소가 문자여야 하기 때문에 다른 시퀀스보다 더 제약된다. 문자열은 또한 불변(immutable)이다. 새로운 문자열을 생성하는 것과 반대로, 문자열의 문자를 변경하고자 한다면, 대신에 문자 리스트를 사용할 필요가 있다.

리스트는 좀더 튜플보다 일반적인다. 부분적으로는 변경가능(mutable)하기 때문이다. 하지만, 튜플을 좀더 선호해야 하는 몇가지 경우가 있다.

1. 어떤 맥락에서 `return`문처럼, 리스트보다 튜플을 생성하는 것이 구문론적으로 간략하다. 다른 맥락에서는 리스트가 더 선호될 수 있다.

2. 딕셔너리 키로서 시퀀스를 사용하려면, 튜플이나 문자열같은 불변형(immutable type)을 사용해야 한다.

3. 함수에 인자로 시퀀스를 전달하려면, 튜플을 사용하는 것이 에일리어싱(aliasing)으로 생기는 예기치 못한 행동에 대한 가능성을 줄인다.

튜플은 불변(immutable)이어서, 현재 리스트를 변경하는 `sort`, `reverse` 같은 메쏘드를 제공하지는 않는다. 하지만, 파이썬은 내장함수 `sorted`, `reversed`를 제공해서, 매개 변수로 임의의 시퀀스를 받아 같은 요소를 다른 순서로 된 새로운 리스트를 반환한다.

1.9 디버깅

리스트, 딕셔너리, 튜플은 **자료 구조(data structures)**로 일반적으로 알려져 있다. 이번장에서 리스트 튜플, 키로 튜플, 값으로 리스트를 담고 있는 딕셔너리 같은 복합 자료 구조를 보기 시작했다. 복합 자료 구조는 유용하지만, **모양 오류(shape errors)**라고 불리는 오류에 노출되어 있다. 즉, 자료 구조가 잘못된 형(type), 크기, 구성일 경우 오류가 발생한다. 혹은 코드를 작성하고, 자료의 모양을 잊게 되면 오류가 발생한다.

예를 들어, 정수 하나인 리스트를 기대하고, 리스트가 아닌 일반 정수를 준다면, 작동하지 않을 것이다.

프로그램을 디버깅할 때, 정말 어려운 버그에 작업을 한다면, 다음 네가지를 시도할 수 있다.

코드 읽기(reading): 코드를 면밀히 조사하고, 반복적으로 읽고, 의도한 해도 프로그램이 작성되었는지를 확인하라.

실행(running): 변경하고, 다른 버젼을 실행해서 실험하라. 종종, 프로그램의 적절한 장소에 적절한 것을 배치해서, 문제가 명확하지만, 때때로, 발판(scaffolding)을 만들기 위해서 많은 시간을 쓰기도 한다.

반추(ruminating): 생각의 시간을 가지세요. 어떤 종류의 오류인가? 구문, 실행, 시맨틱(의미론). 오류 메시지로부터 혹은 프로그램 출력으로부터 무슨 정보를 얻을 수 있는가? 어떤 종류의 오류가 지금 보고 있는 문제를 만들었을까? 문제가 나타나기 전에 마지막으로 바꾼 것은 무엇인가?

퇴각(retreating): 어느 시점에선가, 최선은 물러서서, 최근의 변경을 다시 원복하는 것이다. 동작하고 이해하는 프로그램으로 다시 돌아가서, 다시 프로그램을 작성하는 것이다.

초보 프로그래머는 종종 이들 활동중 하나에 사로잡혀 다른 것을 잊곤 한다. 각 활동은 그 자신만의 실패 양태가 있다.

예를 들어, 프로그램을 정독하는 것은 문제가 인쇄상의 오류에 있다면 도움이 되지만, 문제가 개념상 오해에 뿌리를 두고 있다면 그렇지는 못할 것이다. 만약 여러분이 작성한 프로그램을 이해하지 못한다면, 100번 읽을 수는 있지만, 오류를 발견할 수는 없다. 왜냐하면, 오류는 여러분의 손에 있기 때문입니다.

실험을 수행하는 것은 특히 작고 간단한 테스트를 진행한다면 도움이 될 수 있다. 하지만, 코드를 읽거나, 생각없이 실험을 수행한다면, 프로그램이 작동될 때까지 랜덤 변경을 개발하는 "랜덤 워크 프로그램(random walk programming)" 패턴에 빠질 수 있다. 말할 필요없이 랜덤 워크 프로그래밍은 시간이 오래 걸린다.

생각의 시간을 가져야 한다. 디버깅은 실험 과학 같은 것이다. 문제가 무엇인지에 대한 최소한 한가지 가설을 가져야 한다. 만약 두개 혹은 그 이상의 가능성이 있다면, 이러한 가능성 중에서 하나라도 줄일 수 있는 테스트를 생각해야 한다.

휴식 시간을 가지는 것은 생각하는데 도움이 된다. 대화를 하는 것도 도움이 된다. 문제를 다른 사람 혹은 자신에게도 설명할 수 있다면, 질문을 마치기도 전에 답을 종종 발견할 수 있다.

하지만, 오류가 너무 많고 수정하려는 코드가 너무 많고, 복잡하다면 최고의 디버깅 기술도 무용지물이다. 가끔, 최선의 선택은 퇴각하는 것이다. 작동하고 이해하는 곳까지 후퇴해서 프로그램을 간략화하라.

초보 프로그래머는 종종 퇴각하기를 꺼려한다. 왜냐하면, 설사 틀렸지만, 몇 라인의 코드를 지울 수 없기 때문이다. 삭제하지 않는 것이 기분이 좋다면, 프로그램을 다시 작성하기 전에 프로그램을 다른 파일에 복사하라. 그리고 나서, 한번에 조금씩 붙여넣어라.

정말 어려운 버그(hard bug)를 발견하고 고치는 것은 코드 일기, 실행, 반추, 때때로 퇴각을 요구한다. 만약 이들 활동 중 하나에 빠져있다면, 다른 것들을 시도해보세요.

1.10 용어정의

comparable: A type where one value can be checked to see if it is greater than, less than or equal to another value of the same type. Types which are comparable can be put in a list and sorted.

data structure: A collection of related values, often organized in lists, dictionaries, tuples, etc.

DSU: Abbreviation of "decorate-sort-undecorate," a pattern that involves building a list of tuples, sorting, and extracting part of the result.

gather: The operation of assembling a variable-length argument tuple.

hashable: A type that has a hash function. Immutable types like integers, floats and strings are hashable; mutable types like lists and dictionaries are not.

scatter: The operation of treating a sequence as a list of arguments.

shape (of a data structure): A summary of the type, size and composition of a data structure.

singleton: A list (or other sequence) with a single element.

tuple: An immutable sequence of elements.

tuple assignment: An assignment with a sequence on the right side and a tuple of variables on the left. The right side is evaluated and then its elements are assigned to the variables on the left.

1.11 Exercises

Exercise 1.1 Revise a previous program as follows: Read and parse the "From" lines and pull out the addresses from the line. Count the number of messages from each person using a dictionary.

After all the data has been read print the person with the most commits by creating a list of (count, email) tuples from the dictionary and then sorting the list in reverse order and print out the person who has the most commits.

```
Sample Line:
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

Enter a file name: mbox-short.txt
cwen@iupui.edu 5

Enter a file name: mbox.txt
zqian@umich.edu 195
```

Exercise 1.2 This program counts the distribution of the hour of the day for each of the messages. You can pull the hour from the "From" line by finding the time string and then splitting that string into parts using the colon character. Once you have accumulated the counts for each hour, print out the counts, one per line, sorted by hour as shown below.

```
Sample Execution:
python timeofday.py
Enter a file name: mbox-short.txt
04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1
```

Exercise 1.3 Write a program that reads a file and prints the *letters* in decreasing order of frequency. Your program should convert all the input to lower case and only count the letters a-z. Your program should not count spaces, digits, punctuation or anything other than the letters a-z. Find text samples from several different languages and see how letter frequency varies between languages. Compare your results with the tables at `wikipedia.org/wiki/Letter_frequencies`.