Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 6 – 17, 2020

Python for Data Analytics

Tuples

Tuples

- Ordered collection of arbitrary objects
- Accessed by offset
- Immutable sequence
- Fixed-length, heterogeneous, and arbitrarily nestable

```
menu = (1, 2, 5, 9)
a = 1, 2, 5, 9
b = (0, 'ham', 3.14, 99)
c = ('a', ('x', 'y'), 'z')
emptytuple = ()
```

Tuples are like Lists

- Another kind of "sequence"
- Elements are indexed starting at 0

```
>>> num = (4, 1, 9)
>>> print(num[2])
9
>>> print(len(num))
4
>>> print(max(num))
9
>>> print(min(num))
1
```

```
>>> for i in num:
...     print(i)
4
1
9
>>> print(num + ('a', 'b'))
(4, 1, 9, 'a', 'b')
>>> print(num * 2)
(4, 1, 9, 4, 1, 9)
```

Tuples are Immutable

- Unlike a list, once you create a tuple, you cannot alter its contents
- Similar to a string

Lists

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
[9, 8, 6]
```

Strings

```
>>> s = 'ABC'
>>> s[2] = 'D'
Traceback (most recent
call last):
  File "<stdin>", line 1,
in <module>
TypeError: 'str' object
does not support item
assignment
```

Tuples

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback (most recent
call last):
  File "<stdin>", line 1,
in <module>
TypeError: 'tuple' object
does not support item
assignment
```

Things not to do with Tuples

```
>>> x = (4, 1, 9, 0)
>>> x.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'reverse'
```

Why Tuples?

- Python does not have to build tuple structures to be modifiable
- Simpler and more efficient in terms of memory use and performance than lists
- When we are making "temporary variables", we prefer tuples over lists

Tuples and Assignments

- We can also put a tuple on the left-hand side of an assignment statement
- We can even omit the parentheses
- Can be used to return multiple values in a function

```
>>> (x, y) = (4, 'spam')
>>> print(x)
4
>>> y = 1
>>> x, y = y, x
>>> print(x, y)
1 4
```

```
def ret2(a):
    return min(a), max(a)

a, b = ret2([4, 1, 9, 0])
```

Tuples and Dictionaries

- The *dict*.`items()` method in dictionaries returns a list of (key, value) tuples

```
>>> menu = dict()
>>> menu['ham'] = 8.99
>>> menu['egg'] = 0.99
>>> for (k, v) in menu.items():
...     print(k, v)
ham 8.99
egg 0.99
>>> print(menu.items())
dict_items([('ham', 8.99), ('egg', 0.99)])
```

Tuples are Comparable

- The comparison operators work with tuples and other sequences
- If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ('Jones', 'Sally') < ('Jones', 'Sam')
True
>>> ('Jones', 'Sally') > ('Adams', 'Sam')
True
```

Sorting a Dictionary by Keys

- We can use tuples to sort the contents of a dictionary
- First, use *dict*.`items()` to get a list of tuples from the dictionary
- Next, use `sorted()` to sort the list of tuples

```
>>> menu = {'spam': 9.99, 'ham': 8.99, 'egg': 0.99}
>>> print(sorted(menu.items()))
[('egg', 0.99), ('ham', 8.99), ('spam', 9.99)]
>>> for k, v in sorted(menu.items()):
...     print(k, v)
egg 0.99
ham 8.99
spam 9.99
```

Sorting a Dictionary by Values

- Construct a list of tuples of the form (value, key)
- Then, use `sorted()` to sort the list of tuples by value

```
>>> menu = {'spam': 9.99, 'egg': 0.99, 'ham': 8.99}
>>> tmp = list()
>>> for k, v in menu.items():
...     tmp.append((v, k))
>>> print(tmp)
[(9.99, 'spam'), (0.99, 'egg'), (8.99, 'ham')]
>>> tmp = sorted(tmp, reverse=True)
>>> print(tmp)
[(9.99, 'spam'), (8.99, 'ham'), (0.99, 'egg')]
```

The Top 10 Words

```
filename = input('Enter file: ')
f = open(filename)

counts = dict()
for line in f:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

lst = list()
for k, v in counts.items():
    lst.append((v, k))

lst = sorted(lst, reverse=True)

for v, k in lst[:10]:
    print(v, k)
```

```
Enter file: genesis.txt
3630 and
2458 the
1361 of
652 his
644 he
608 to
597 unto
589 in
512 that
470 i
```

Even Shorter Version

- List comprehension creates a dynamic list

```
filename = input('Enter file: ')
f = open(filename)

counts = dict()
for line in f:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

lst = sorted([(v,k) for k,v in counts.items()], reverse=True)

for v, k in lst[:10]:
    print(v, k)
```

Summary

	String	List	Tuple	Dictionary	Set
Initialization	`r = str()` `r = ''`	`l = list()` `l = []`	`t = tuple()` `t = ()`	`d = dict()` `d = {}`	`s = set()`
Example	`r = '123'`	`l = [1, 2, 3]`	`t = (1, 2, 3)`	`d = {1:'a', 2:'b'}`	`s = {1, 2, 3}`
Category	Sequence	Sequence	Sequence	Collection	Collection
Mutable?	No	Yes	No	Yes	Yes
Items ordered?	Yes	Yes	Yes	No	No
Indexing/slicing	Yes	Yes	Yes	No	No
Duplicate items?	Yes	Yes	Yes	No (unique key)	No
Items sorted?	No	No	No	No	No
`in` operator	Yes	Yes	Yes	Yes	Yes