TUPLE

Overview

- A tuple is a sequence of values.
- The values can be any type, and they are indexed by integers, so in that respect tuples are a lot like lists.
- The important difference is that tuples are immutable.

Initializing

- 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:

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

Elements

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

  t = tuple(‘PYTHON’)

- print t

  (‘P', ‘Y', ‘T', ‘H', ‘O', ‘N')

Lists and Tuples

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

```
t = (‘P', ‘Y', ‘T', ‘H’, ‘O’, ‘N’ )
print t[3]
‘H’
print t[1:3]
( ‘Y’, ‘T’ )
```

Return Values

- A function can only return one value, but if the value is a tuple, the effect is the same as returning multiple values.
- For example, if you want to divide two integers and compute the quotient and remainder, it is inefficient to compute x/y and then x%y.
- It is better to compute them both at the same time.

```
t = (7/3, 7%3)
print( t )
prints:  ( 2, 1 )
```

ZIP

- Zip is a built-in function that takes two or more sequences and “zips” them into a list of tuples where each tuple contains one element from each sequence. This example zips a string and a list:

```
x = 'abc’
y = [0, 1, 2]
zip(x, y)
[ ('a', 0), ('b', 1), ('c', 2) ]
```

Dictionaries

- Dictionaries have a method called items that returns a list of tuples, where each tuple is a key-value pair.

```
d = {'a':0, 'b':1, 'c':2}
t = d.items()
print t
[('a', 0), ('c', 2), ('b', 1)]
```

Comparing

- The relational operators work with tuples and other sequences; Python starts by comparing the first element from each sequence. If they are equal, it goes on to the next elements, and so on, until it finds elements that differ. Subsequent elements are not considered (even if they are larger).

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

Sorting

```
def sort_by_length(words):  #words is a read-only tuple
        t = []
        for word in words:
                t.append((len(word), word))
        t.sort(reverse=True)
        res = []
        for length, word in t:
                res.append((length,word))
        return res
```