Tuples

- Tuples construct simple groups of objects. They work exactly like lists, except that tuples can't be changed in place (they're immutable)
- Tuples are:
  - *Ordered collections of arbitrary objects*
  - *Accessed by offset*
  - *Of the category "immutable sequence"*
  - *Fixed-length, heterogeneous, and arbitrarily nestable*
  - *Arrays of object references*

```
>>> (1, 2) + (3, 4)          # Concatenation
(1, 2, 3, 4)

>>> (1, 2) * 4               # Repetition
(1, 2, 1, 2, 1, 2, 1, 2)

>>> T = (1, 2, 3, 4)         # Indexing, slicing
>>> T[0], T[1:3]
(1, (2, 3))
```


Conversions, methods, and immutability

- Tuples don't provide the same methods you saw for strings, lists, and dictionaries
- E.g. for sort, you have to convert tuple to list or use built-in sorted

```
>>> T = ('cc', 'aa', 'dd', 'bb')
>>> tmp = list(T)             # Make a list from a tuple's items
>>> tmp.sort()                # Sort the list
>>> tmp
['aa', 'bb', 'cc', 'dd']
>>> T = tuple(tmp)            # Make a tuple from the list's items
>>> T
('aa', 'bb', 'cc', 'dd')

>>> sorted(T)                 # Or use the sorted built-in, and save two steps
['aa', 'bb', 'cc', 'dd']
```


Conversions, methods, and immutability

- ***index*** and ***count*** work as they do for lists, but they are defined for tuple objects:

```
>>> T = (1, 2, 3, 2, 4, 2)      # Tuple methods in 2.6, 3.0, and later
>>> T.index(2)                  # Offset of first appearance of 2
1
>>> T.index(2, 2)               # Offset of appearance after offset 2
3
>>> T.count(2)                  # How many 2s are there?
3
```

- The rule about tuple *immutability* applies only to the top level of the tuple itself, not to its contents. A list inside a tuple, for instance, can be changed as usual

```
>>> T = (1, [2, 3], 4)
>>> T[1] = 'spam'               # This fails: can't change tuple itself
TypeError: object doesn't support item assignment

>>> T[1][0] = 'spam'            # This works: can change mutables inside
>>> T
(1, ['spam', 3], 4)
```


Comparing tuples

- The comparison 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 element, and so on, until it finds elements that differ. Subsequent elements are not considered (even if they are really big)

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


Why Lists and Tuples?

- The immutability of tuples provides some *integrity*—you can be sure a tuple won't be changed through another reference elsewhere in a program, but there's no such guarantee for lists
- Tuples can also be used in places that lists cannot—for example, as dictionary keys