>>> t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')

>>> t

('foo', 'bar', 'baz', 'qux', 'quux', 'corge')

>>> t[0]

'foo'

>>> t[-1]

'corge'

>>> t[1::2]

('bar', 'qux', 'corge')

>>> t[::-1]

('corge', 'quux', 'qux', 'baz', 'bar', 'foo')

Everything you’ve learned about lists—they are ordered, they can contain arbitrary objects, they can be indexed and sliced, they can be nested—is true of tuples as well. But they can’t be modified:

>>> t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')

>>> t[2] = 'Bark!'

Traceback (most recent call last):

File "<pyshell#65>", line 1, in <module>

t[2] = 'Bark!'

TypeError: 'tuple' object does not support item assignment

Why use a tuple instead of a list?

Program execution is faster when manipulating a tuple than it is for the equivalent list

Sometimes you don’t want data to be modified.

>>> a = 'foo'

>>> b = 42

>>> a, 3.14159, b

('foo', 3.14159, 42)

But what happens when you try to define a tuple with one item:

>>> t = (2)

>>> type(t)

<class 'int'>

To tell Python that you really want to define a singleton tuple, include a trailing comma (,) just before the closing parenthesis:

>>> t = (2,)

>>> type(t)

<class 'tuple'>

>>> t[0]

2

>>> t[-1]

2

Tuple Assignment, Packing, and Unpacking

>>> t = ('foo', 'bar', 'baz', 'qux')

Tuple Unpacking

>>> (s1, s2, s3, s4) = t

>>> s1

'foo'

>>> s2

'bar'

>>> s3

'baz'

>>> s4

'qux'

Packing and unpacking can be combined into one statement to make a compound assignment:

>>> (s1, s2, s3, s4) = ('foo', 'bar', 'baz', 'qux')

>>> s1

'foo'

>>> s2

'bar'

>>> s3

'baz'

>>> s4

'qux'