CME 193: Introduction to Scientific Python

Winter 2017

Lecture 3: Containers

Blake Jennings

`stanford.edu/~bmj/cme193`

Contents

Tuples

Seemingly similar to lists

```
>>> myTuple = (1, 2, 3)
>>> myTuple[1]
2
>>> myTuple[1:3]
(2, 3)
```

Tuples are immutable

Unlike lists, we cannot change elements.

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

Packing and unpacking

```
t = 1, 2, 3
x, y, z = t

print t  # (1, 2, 3)
print y  # 2
```

Functions with multiple return values

```
def simple_function():
    return 0, 1, 2

print simple_function()
# (0, 1, 2)
```

Contents

Dictionaries

A dictionary is a *collection* of *key-value* pairs.

An example: the keys are all words in the English language, and their corresponding values are the meanings.

No sense of order for the dictionary, just key-value pairs.

Denoted by curly brackets `{}`

Defining a dictionary

```
>>> d = {}
>>> d[1] = "one"
>>> d[2] = "two"
>>> d
{1: 'one', 2: 'two'}
>>> e = {1: 'one', 'hello': True}
>>> e
{1: 'one', 'hello': True}
```

Note how we can add more key-value pairs at any time. Also, only condition on keys is that they are *immutable*.

No duplicate keys

Old value gets overwritten instead!

```
>>> d = {1: 'one', 2: 'two'}
>>> d[1] = 'three'
>>> d
{1: 'three', 2: 'two'}
```

Access

We can access values by keys, but not the other way around

```
>>> d = {1: 'one', 2: 'two'}
>>> print d[1]
```

Furthermore, we can check whether a key is in the dictionary by

`key in dict`

Access

We can access values by keys, but not the other way around

```
>>> d = {1: 'one', 2: 'two'}
>>> print d[1]
```

Furthermore, we can check whether a key is in the dictionary by

`key in dict`

All keys, values or both

Use `d.keys()`, `d.values()` and `d.items()`

```
>>> d = {1: 'one', 2: 'two', 3: 'three'}
>>> d
{1: 'one', 2: 'two', 3: 'three'}
>>> d.keys()
[1, 2, 3]
>>> d.values()
['one', 'two', 'three']
>>> d.items()
[(1, 'one'), (2, 'two'), (3, 'three')]
```

So how can you loop over dictionaries?

Small exercise

Print all key-value pairs of a dictionary

```
>>> d = {1: 'one', 2: 'two', 3: 'three'}
>>> for key, value in d.items():
...     print key, value
...
1 one
2 two
3 three
```

Instead of `d.items()`, you can use `d.iteritems()` as well. Better performance for large dictionaries. Or can write `for key in d` to just iterate over keys

Small exercise

Print all key-value pairs of a dictionary

```
>>> d = {1: 'one', 2: 'two', 3: 'three'}
>>> for key, value in d.items():
...     print key, value
...
1 one
2 two
3 three
```

Instead of `d.items()`, you can use `d.iteritems()` as well. Better performance for large dictionaries. Or can write `for key in d` to just iterate over keys

Contents

Sets

Sets are an unordered collection of unique elements

```
>>> basket = ['apple', 'orange', 'apple',
    'pear', 'orange', 'banana']
>>> fruit = set(basket)  # create a set
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit  # fast membership testing
True
>>> 'crabgrass' in fruit
False
```

Implementation: like a dictionary only keys.

from: Python documentation

Set comprehensions

```
>>> a = {x for x in ’abracadabra’ if x not in ’abc’}
>>> a
set([’r’, ’d’])
```

from: Python documentation

Contents

Strings

Let's quickly go over *strings*.

- Strings hold a sequence of characters.
- Strings are *immutable*
- We can slice strings just like lists and tuples
- Between quotes or triple quotes
- Why might you want to use triple quotes?

Everything can be turned into a string!

We can turn anything in Python into a string using `str`.

This includes dictionaries, lists, tuples, etc.

String formatting

- Special characters: \n, \t, etc
- Add variables: %s, %f, %e, %g, %d, or use `format`

```
fl = 0.23
wo = 'Hello'
inte = 12

print "s: {} \t f: {:0.1f} \n i: {}".format(wo, fl, inte)
# s: Hello     f: 0.2
#  i: 12
```

String formatting

```
fl = 0.23
wo = 'Hello'
inte = 12

print "s: %s \t f: %.1f \n i: %d" % (wo, fl, inte)
# s: Hello     f: 0.2
#  i: 12
```

See documentation for many more options and examples!

Split

To split a string, for example, into seperate words, we can use `split()`

```
text = 'Hello, world!\n How are you?'
text.split()
# ['Hello,', 'world!', 'How', 'are', 'you?']
```

Split

What if we have a comma seperated file with numbers seperated by commas?

```
numbers = '1, 3, 2, 5'
numbers.split()
# ['1,', '3,', '2,', '5']

numbers.split(', ')
# ['1', '3', '2', '5']

[int(i) for i in numbers.split(', ')]
# [1, 3, 2, 5]
```

Use the optional argument in `split()` to use a custom seperator.

UPPER and lowercase

There are a bunch of useful string functions, such as `.lower()` and `.upper()` that turn your string in lower- and uppercase.

Note: To quickly find all functions for a string, we can use `dir`

```
text = 'hello'
dir(text)
```

join

Another handy function: `join`.

We can use join to create a string from a list.

```
words = ['hello', 'world']
' '.join(words)

''.join(words)
# 'helloworld'

' '.join(words)
# 'hello world'

', '.join(words)
# 'hello, world'
```

Input must be a list or tuple of strings!