2-Data

December 9, 2014

0.1 Tuples

- Think of them as a C-struct: associating a number of objects together
- immutable: once created, references in tuple instance cannot be changed:
  - NOTE: this doesn't mean that the objects *inside* the tuple cannot have their state changed, but that depends on those contained objects mutability
- ordered with index lookup
- no constraints on what is contained in the tuple
  - any object
  - no uniqueness constraint

```
In [1]: # tuple version of points:
        a = (3, 4)
        b = (9, 6)

        from math import sqrt
        def dist(p1, p2):
            ' tuple version of point distance '
            return sqrt((p2[0] - p1[0])**2 + (p2[1] - p1[1])**2)

In [2]: dist(a, b)

Out[2]: 6.324555320336759

In [4]: # dict version of points:
        a = dict(x=3, y=4)
        b = dict(x=9, y=6)

        def dist(p1, p2):
            ' tuple version of point distance '
            return sqrt((p2['x'] - p1['x'])**2 + (p2['y'] - p1['y'])**2)

In [5]: a

Out[5]: {'y': 4, 'x': 3}

In [6]: b

Out[6]: {'y': 6, 'x': 9}

In [7]: dist(a, b)

Out[7]: 6.324555320336759
```

```
In [10]: g = ('GOOG', 100, 530.18)
         h = ('HP', 250, 38.17)
         a = ('AAPL', 50, 112.90)
         stocks = [g, h, a]

In [11]: stocks

Out[11]: [('GOOG', 100, 530.18), ('HP', 250, 38.17), ('AAPL', 50, 112.9)]

In [12]: from collections import namedtuple

In [29]: StockTuple = namedtuple('StockTuple', ['tick', 'count', 'price'], verbose=True)

from builtins import property as _property, tuple as _tuple
from operator import itemgetter as _itemgetter
from collections import OrderedDict

class StockTuple(tuple):
    'StockTuple(tick, count, price)'

    __slots__ = ()

    _fields = ('tick', 'count', 'price')

    def __new__(_cls, tick, count, price):
        'Create new instance of StockTuple(tick, count, price)'
        return _tuple.__new__(_cls, (tick, count, price))

    @classmethod
    def _make(cls, iterable, new=tuple.__new__, len=len):
        'Make a new StockTuple object from a sequence or iterable'
        result = new(cls, iterable)
        if len(result) != 3:
            raise TypeError('Expected 3 arguments, got %d' % len(result))
        return result

    def _replace(_self, **kwds):
        'Return a new StockTuple object replacing specified fields with new values'
        result = _self._make(map(kwds.pop, ('tick', 'count', 'price'), _self))
        if kwds:
            raise ValueError('Got unexpected field names: %r' % list(kwds))
        return result

    def __repr__(self):
        'Return a nicely formatted representation string'
        return self.__class__.__name__ + '(tick=%r, count=%r, price=%r)' % self

    @property
    def __dict__(self):
        'A new OrderedDict mapping field names to their values'
        return OrderedDict(zip(self._fields, self))

    def _asdict(self):
        'Return a new OrderedDict which maps field names to their values.'
        return self.__dict__
    def __getnewargs__(self):
        'Return self as a plain tuple.  Used by copy and pickle.'
        return tuple(self)

    def __getstate__(self):
        'Exclude the OrderedDict from pickling'
        return None

    tick = _property(_itemgetter(0), doc='Alias for field number 0')

    count = _property(_itemgetter(1), doc='Alias for field number 1')

    price = _property(_itemgetter(2), doc='Alias for field number 2')
```

```
In [14]: StockTuple

Out[14]: __main__.StockTuple

In [15]: g = StockTuple('GOOG', 100, 530.18)
         h = StockTuple('HP', 250, 38.17)
         a = StockTuple('AAPL', 50, 112.90)

In [16]: stocks = [g, h, a]

In [17]: g

Out[17]: StockTuple(tick='GOOG', count=100, price=530.18)

In [18]: h

Out[18]: StockTuple(tick='HP', count=250, price=38.17)

In [19]: a

Out[19]: StockTuple(tick='AAPL', count=50, price=112.9)

In [20]: h[0]

Out[20]: 'HP'

In [21]: h[1]

Out[21]: 250

In [22]: h[2]

Out[22]: 38.17

In [24]: h[2] = 45.25 # we can't update fields
```

```
        ---------------------------------------------------------------------------
    TypeError                                 Traceback (most recent call last)

        <ipython-input-24-da20b11080b8> in <module>()
    ----> 1 h[2] = 45.25 # we can't update fields


        TypeError: 'StockTuple' object does not support item assignment
```

```
In [25]: h.tick

Out[25]: 'HP'

In [26]: h.count

Out[26]: 250

In [27]: h.price

Out[27]: 38.17

In [28]: stocks

Out[28]: [StockTuple(tick='GOOG', count=100, price=530.18),
          StockTuple(tick='HP', count=250, price=38.17),
          StockTuple(tick='AAPL', count=50, price=112.9)]
```

0.2 Lists

- mutable collections of the same kind of thing
- mutable (add, remove, change)
- ordered
- no uniqueness constraint (reference to same object can occur multiple times)

```
In [30]: nums = [10, 20, 30]
         stuff = ['foo', 'bar', nums, 3.14, nums, 'bang']

In [30]:

In [31]: stuff

Out[31]: ['foo', 'bar', [10, 20, 30], 3.14, [10, 20, 30], 'bang']

In [32]: nums.append(40)

In [33]: nums

Out[33]: [10, 20, 30, 40]

In [34]: stuff

Out[34]: ['foo', 'bar', [10, 20, 30, 40], 3.14, [10, 20, 30, 40], 'bang']

In [35]: stuff[2]

Out[35]: [10, 20, 30, 40]

In [36]: stuff[2].append(50)

In [37]: nums

Out[37]: [10, 20, 30, 40, 50]

In [38]: stuff

Out[38]: ['foo', 'bar', [10, 20, 30, 40, 50], 3.14, [10, 20, 30, 40, 50], 'bang']

In [39]: stuff[1]
```

```
Out[39]: 'bar'

In [40]: stuff[1] = 'ping'

In [41]: stuff

Out[41]: ['foo', 'ping', [10, 20, 30, 40, 50], 3.14, [10, 20, 30, 40, 50], 'bang']

In [42]: meta = 'foo bar zip zap ping pong'.split()

In [43]: meta

Out[43]: ['foo', 'bar', 'zip', 'zap', 'ping', 'pong']

In [44]: sorted(meta)

Out[44]: ['bar', 'foo', 'ping', 'pong', 'zap', 'zip']

In [45]: meta

Out[45]: ['foo', 'bar', 'zip', 'zap', 'ping', 'pong']

In [48]: reversed(meta)

Out[48]: <list_reverseiterator at 0x107421438>

In [47]: list(reversed(meta))

Out[47]: ['pong', 'ping', 'zap', 'zip', 'bar', 'foo']

In [49]: meta

Out[49]: ['foo', 'bar', 'zip', 'zap', 'ping', 'pong']

In [50]: meta.reverse() # method on meta

In [51]: meta

Out[51]: ['pong', 'ping', 'zap', 'zip', 'bar', 'foo']

In [52]: meta.sort()

In [53]: meta

Out[53]: ['bar', 'foo', 'ping', 'pong', 'zap', 'zip']

In [54]: meta.sort(reverse=True)

In [55]: meta

Out[55]: ['zip', 'zap', 'pong', 'ping', 'foo', 'bar']

In [72]: grays = {'black', 'white', 'bone', 'gray', 'midnight'}
         solids = {'black', 'white', 'red', 'green', 'blue'}
         pastels = {'gray', 'pink', 'purple', 'bone'}

In [57]: grays & solids

Out[57]: {'black', 'white'}

In [58]: grays | solids
```

```
Out[58]: {'black', 'blue', 'bone', 'gray', 'green', 'midnight', 'red', 'white'}

In [73]: grays & pastels

Out[73]: {'bone', 'gray'}

In [64]: words = set('foo bar ping bar zip pow zap bar foo ping pow'.split())

In [65]: words

Out[65]: {'bar', 'foo', 'ping', 'pow', 'zap', 'zip'}

In [66]: %pprint

Pretty printing has been turned OFF

In [67]: words

Out[67]: {'foo', 'bar', 'zip', 'pow', 'zap', 'ping'}

In [68]: words.update("blort ping bang wibble zip zap pow".split())

In [69]: words

Out[69]: {'bang', 'blort', 'foo', 'bar', 'zip', 'wibble', 'pow', 'zap', 'ping'}

In [70]: grays - solids

Out[70]: {'bone', 'gray', 'midnight'}

In [74]: pastels - grays

Out[74]: {'pink', 'purple'}

In [75]: help(grays)

Help on set object:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __isub__(self, value, /)
 |      Return self-=value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __ixor__(self, value, /)
 |      Return self^=value.
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  __or__(self, value, /)
 |      Return self|value.
 |
 |  __rand__(self, value, /)
 |      Return value&self.
 |
 |  __reduce__(...)
 |      Return state information for pickling.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __ror__(self, value, /)
 |      Return value|self.
 |
 |  __rsub__(self, value, /)
 |      Return value-self.
 |
 |  __rxor__(self, value, /)
 |      Return value^self.
 |
 |  __sizeof__(...)
 |      S.__sizeof__() -> size of S in memory, in bytes
 |
 |  __sub__(self, value, /)
 |      Return self-value.
 |
 |  __xor__(self, value, /)
 |      Return self^value.
 |
 |  add(...)
 |      Add an element to a set.
 |
 |      This has no effect if the element is already present.
 |
 |  clear(...)
 |      Remove all elements from this set.
 |
 |  copy(...)
 |      Return a shallow copy of a set.
 |
 |  difference(...)
 |      Return the difference of two or more sets as a new set.
 |
 |      (i.e. all elements that are in this set but not the others.)
 |
 |  difference_update(...)
 |      Remove all elements of another set from this set.
 |
 |  discard(...)
 |      Remove an element from a set if it is a member.
 |
 |      If the element is not a member, do nothing.
 |
 |  intersection(...)
 |      Return the intersection of two sets as a new set.
 |
 |      (i.e. all elements that are in both sets.)
 |
 |  intersection_update(...)
 |      Update a set with the intersection of itself and another.
 |
 |  isdisjoint(...)
 |      Return True if two sets have a null intersection.
 |
 |  issubset(...)
 |      Report whether another set contains this set.
 |
 |  issuperset(...)
```

```
 |      Report whether this set contains another set.
 |
 |  pop(...)
 |      Remove and return an arbitrary set element.
 |      Raises KeyError if the set is empty.
 |
 |  remove(...)
 |      Remove an element from a set; it must be a member.
 |
 |      If the element is not a member, raise a KeyError.
 |
 |  symmetric_difference(...)
 |      Return the symmetric difference of two sets as a new set.
 |
 |      (i.e. all elements that are in exactly one of the sets.)
 |
 |  symmetric_difference_update(...)
 |      Update a set with the symmetric difference of itself and another.
 |
 |  union(...)
 |      Return the union of sets as a new set.
 |
 |      (i.e. all elements that are in either set.)
 |
 |  update(...)
 |      Update a set with the union of itself and others.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

```
In [76]: help(meta)
```

```
Help on list object:

class list(object)
 |  list() -> new empty list
 |  list(iterable) -> new list initialized from iterable's items
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(...)
|      x.__getitem__(y) <==> x[y]
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __iadd__(self, value, /)
|      Implement self+=value.
|
|  __imul__(self, value, /)
|      Implement self*=value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.n
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(...)
|      L.__reversed__() -- return a reverse iterator over the list
|
|  __rmul__(self, value, /)
|      Return self*value.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(...)
```

```
 |      L.__sizeof__() -- size of L in memory, in bytes
 |
 |  append(...)
 |      L.append(object) -> None -- append object to end
 |
 |  clear(...)
 |      L.clear() -> None -- remove all items from L
 |
 |  copy(...)
 |      L.copy() -> list -- a shallow copy of L
 |
 |  count(...)
 |      L.count(value) -> integer -- return number of occurrences of value
 |
 |  extend(...)
 |      L.extend(iterable) -> None -- extend list by appending elements from the iterable
 |
 |  index(...)
 |      L.index(value, [start, [stop]]) -> integer -- return first index of value.
 |      Raises ValueError if the value is not present.
 |
 |  insert(...)
 |      L.insert(index, object) -- insert object before index
 |
 |  pop(...)
 |      L.pop([index]) -> item -- remove and return item at index (default last).
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(...)
 |      L.remove(value) -> None -- remove first occurrence of value.
 |      Raises ValueError if the value is not present.
 |
 |  reverse(...)
 |      L.reverse() -- reverse *IN PLACE*
 |
 |  sort(...)
 |      L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

```
In []:
```