2-Data

December 10, 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
```

0.3 List Slicing

```
In [99]: nums = [3, 5, 2, 7, 5, 5, 6, 3, 1]

In [101]: nums[0]

Out[101]: 3

In [102]: # slice: start:end
          nums[2:5] # half open interval: from 2 up to but not incl 5

Out[102]: [2, 7, 5]
```

```
In [103]: nums[2], nums[3], nums[4]

Out[103]: (2, 7, 5)

In [104]: # can leave off start, defaults to 0
          nums[:5]

Out[104]: [3, 5, 2, 7, 5]

In [105]: # can leave off the end, defaults to len(list)
          nums[4:]

Out[105]: [5, 5, 6, 3, 1]

In [109]: part = nums[4:]

In [111]: id(nums)

Out[111]: 4416693896

In [112]: nums

Out[112]: [3, 5, 2, 7, 5, 5, 6, 3, 1]

In [114]: part

Out[114]: [5, 5, 6, 3, 1]

In [116]: id(part) # same or different from nums?

Out[116]: 4416726152

In [119]: nums[4]

Out[119]: 5

In [118]: id(nums[4])

Out[118]: 4297326752

In [121]: id(part[0])

Out[121]: 4297326752

In [122]: part[0]

Out[122]: 5

In []: # slices return new lists with a shallow copy from the original list

In [106]: nums[:]

Out[106]: [3, 5, 2, 7, 5, 5, 6, 3, 1]

In [107]: result = nums[:]

In [123]: copy = nums # we already know that this is just another alias to nums
                      # two identical references to the same list

In [124]: copy is nums
```

```
Out[124]: True

In [125]: id(nums)

Out[125]: 4416693896

In [126]: id(result)

Out[126]: 4416720584

In [127]: result is nums

Out[127]: False

In [128]: a = 1234

In [129]: b = 1200 + 34

In [130]: id(a)

Out[130]: 4417193392

In [131]: id(b)

Out[131]: 4417193360

In [132]: a == b

Out[132]: True

In [133]: a is b

Out[133]: False

In [134]: x = 42
          y = 40 + 2

In [135]: x == y

Out[135]: True

In [136]: x is y # only because CPython interns -5 to 255 so there is only one
                 # instance of each of these integers

Out[136]: True

In [137]: nums

Out[137]: [3, 5, 2, 7, 5, 5, 6, 3, 1]

In [141]: nums[0:45:3] # strides

Out[141]: [3, 7, 6]

In [142]: nums[:100000] # this is a surprise to most people

Out[142]: [3, 5, 2, 7, 5, 5, 6, 3, 1]

In [143]: nums[7:2:-1]

Out[143]: [3, 6, 5, 5, 7]
```

```
In [144]: nums[7:2:-2]

Out[144]: [3, 5, 7]

In [145]: nums[::-1]

Out[145]: [1, 3, 6, 5, 5, 7, 2, 5, 3]

In [161]: meta = "foo bar zip".split()

In [162]: meta

Out[162]: ['foo', 'bar', 'zip']

In [163]: meta[1]

Out[163]: 'bar'

In [164]: meta[1] = 'baz'

In [165]: meta

Out[165]: ['foo', 'baz', 'zip']

In [166]: meta.append('zap')

In [167]: meta

Out[167]: ['foo', 'baz', 'zip', 'zap']

In [168]: extra = "blort wibble".split()

In [169]: extra

Out[169]: ['blort', 'wibble']

In [170]: meta + extra

Out[170]: ['foo', 'baz', 'zip', 'zap', 'blort', 'wibble']

In [172]: meta

Out[172]: ['foo', 'baz', 'zip', 'zap']

In [171]: id(meta)

Out[171]: 4417107272

In [173]: new = meta + extra

In [174]: id(new)

Out[174]: 4417109768

In [175]: meta

Out[175]: ['foo', 'baz', 'zip', 'zap']

In [176]: meta = meta + extra

In [177]: id(meta)
```

```
Out[177]: 4417107144

In [158]: meta += extra

In [159]: meta

Out[159]: ['foo', 'baz', 'zip', 'zap', 'blort', 'wibble']

In [160]: id(meta)

Out[160]: 4417071304

In [178]: meta = meta + extra

In [188]: ianshopping = 'milk bread butter'.split()

In [189]: emilyshopping = ianshopping

In [190]: otherstuff = 'cereal oj jam'.split()

In [191]: ianshopping += otherstuff # ianshopping.__inc__(otherstuff)

In [192]: ianshopping

Out[192]: ['milk', 'bread', 'butter', 'cereal', 'oj', 'jam']

In [193]: emilyshopping

Out[193]: ['milk', 'bread', 'butter', 'cereal', 'oj', 'jam']

In [194]: ianshopping

Out[194]: ['milk', 'bread', 'butter', 'cereal', 'oj', 'jam']

In [195]: 'oj' in ianshopping

Out[195]: True

In [197]: 'OJ' in ianshopping

Out[197]: False

In [198]: # my own case insensitive search
          item = 'Oj'
          item.lower() in [i.lower() for i in ianshopping]

Out[198]: True

In [199]: ianshopping

Out[199]: ['milk', 'bread', 'butter', 'cereal', 'oj', 'jam']

In [200]: 'read' in ianshopping

Out[200]: False

In [201]: words = 'milk bread butter cereal'

In [203]: len(words) # string, not list

Out[203]: 24
```

```
In [204]: 'ilk' in words

Out[204]: True

In [205]: 'ad' in words

Out[205]: True

In [206]: 'BUTTER' in words

Out[206]: False

In [207]: words.startswith('milk')

Out[207]: True

In [208]: words.split('a')

Out[208]: ['milk bre', 'd butter cere', 'l']

In [209]: words.split('ad')

Out[209]: ['milk bre', ' butter cereal']

In [210]: ianshopping

Out[210]: ['milk', 'bread', 'butter', 'cereal', 'oj', 'jam']

In [211]: ianshopping.pop()

Out[211]: 'jam'

In [212]: ianshopping.pop()

Out[212]: 'oj'

In [213]: ianshopping

Out[213]: ['milk', 'bread', 'butter', 'cereal']

In [214]: extra

Out[214]: ['blort', 'wibble']

In [215]: ianshopping.append(extra) # do you think this is what I want to do?

In [216]: len(ianshopping)

Out[216]: 5

In [217]: ianshopping

Out[217]: ['milk', 'bread', 'butter', 'cereal', ['blort', 'wibble']]

In [218]: ianshopping.pop()

Out[218]: ['blort', 'wibble']

In [219]: ianshopping

Out[219]: ['milk', 'bread', 'butter', 'cereal']
```

```
In [220]: veg = 'carrots squash celery'.split()

In [222]: veg

Out[222]: ['carrots', 'squash', 'celery']

In [223]: ianshopping.extend(veg)

In [224]: ianshopping

Out[224]: ['milk', 'bread', 'butter', 'cereal', 'carrots', 'squash', 'celery']

In [225]: ianshopping.insert(3, 'tomatoes')

In [226]: ianshopping[3]

Out[226]: 'tomatoes'

In [227]: ianshopping

Out[227]: ['milk', 'bread', 'butter', 'tomatoes', 'cereal', 'carrots', 'squash', 'celery']

In [228]: ianshopping.remove('bread')

In [229]: ianshopping

Out[229]: ['milk', 'butter', 'tomatoes', 'cereal', 'carrots', 'squash', 'celery']

In [230]: ianshopping[4]

Out[230]: 'carrots'

In [231]: del ianshopping[4]

In [232]: ianshopping

Out[232]: ['milk', 'butter', 'tomatoes', 'cereal', 'squash', 'celery']

In [233]: nums

Out[233]: [3, 5, 2, 7, 5, 5, 6, 3, 1]

In [234]: nums[2:5]

Out[234]: [2, 7, 5]

In [235]: nums[2:5] = [1,2,3] # make a prediction: what will happen

In [236]: nums

Out[236]: [3, 5, 1, 2, 3, 5, 6, 3, 1]

In [237]: nums[2:5]

Out[237]: [1, 2, 3]

In [238]: nums[2:5] = [10, 20, 30, 40, 50] # make a prediction
          # 1. exception (size mismatch)
          # 2. all "copied", tail shifts right -- WINNER!
          # 3. 40, 50 overwrite 5, 6
          # 4. 40, 50 are ignored
```

```
In [239]: nums

Out[239]: [3, 5, 10, 20, 30, 40, 50, 5, 6, 3, 1]

In [240]: nums[2:7]

Out[240]: [10, 20, 30, 40, 50]

In [241]: nums[2:7] = [888, 999] # what now?
          # 1. exception (mismatch)
          # 2. 10, 20 become 888, 999, rest unchanged
          # 3. repeats over range (inserts [888, 999, 888, 999, 888])
          # 4. 888, 999 replaces specified range -- WINNER!

In [242]: nums

Out[242]: [3, 5, 888, 999, 5, 6, 3, 1]

In [243]: ianshopping

Out[243]: ['milk', 'butter', 'tomatoes', 'cereal', 'squash', 'celery']

In [244]: emilyshopping

Out[244]: ['milk', 'butter', 'tomatoes', 'cereal', 'squash', 'celery']

In [245]: ianshopping is emilyshopping

Out[245]: True

In [246]: ianshopping[:] # in read context, this creates new list with shallow copy
                         # of everything in the original list

Out[246]: ['milk', 'butter', 'tomatoes', 'cereal', 'squash', 'celery']

In [247]: id(ianshopping)

Out[247]: 4417103880

In [265]: ianshopping[:] = 'beer pizza ice_cream'.split()

In [268]: emilyshopping = ianshopping

In [269]: id(ianshopping)

Out[269]: 4417103880

In [270]: id(emilyshopping)

Out[270]: 4417103880

In [271]: ianshopping

Out[271]: ['beer', 'pizza', 'ice_cream']

In [272]: emilyshopping

Out[272]: ['beer', 'pizza', 'ice_cream']
```

```
In [273]: #emilyshopping[:] = [] # then we're not going shopping
          #emilyshopping = [] # haha, my superior Python skills save my
                              #pizza and beer night
          del emilyshopping # take 2 to cancel pizza and beer night

In [274]: id(emilyshopping)


        ---------------------------------------------------------------------------
    NameError                                 Traceback (most recent call last)

        <ipython-input-274-ba910240d106> in <module>()
    ----> 1 id(emilyshopping)


        NameError: name 'emilyshopping' is not defined


In [275]: ianshopping

Out[275]: ['beer', 'pizza', 'ice_cream']
```

0.4 Iteration

Iteration is a core idiom in Python and always has been. With Python 3, it is taken to an even greater level in the reworking of standard APIs around core data structures and libraries.

Iterator Protocol `python it = iter(ITERABLE) while True: try: VALS = next(it) EXPR(VALS) except StopIteration: break`

This is what the for loop does behind the scenes:

```
for VALS in ITERABLE:
    EXPR(VALS)

In [277]: range(5)

Out[277]: range(0, 5)

In [278]: range(3, 8)

Out[278]: range(3, 8)

In [279]: range(12, 55, 11)

Out[279]: range(12, 55, 11)

In [280]: list(range(12, 55, 11)) # turn an iterable into a a reified list

Out[280]: [12, 23, 34, 45]

In [281]: r = range(3, 15, 2)

In [282]: r

Out[282]: range(3, 15, 2)

In [283]: rit = iter(r) # helper function that returns r.__iter__()

In [284]: rit
```

```
Out[284]: <range_iterator object at 0x10589bc00>

In [285]: rit2 = iter(r)

In [286]: rit2
Out[286]: <range_iterator object at 0x10589b8d0>

In [288]: next(rit) # next() is a builtin helper function that calls rit.__next__()
Out[288]: 3

In [289]: next(rit)
Out[289]: 5

In [290]: rit.__next__()
Out[290]: 7

In [291]: rit.__next__()
Out[291]: 9

In [292]: r
Out[292]: range(3, 15, 2)

In [293]: next(rit2)
Out[293]: 3

In [294]: next(rit2)
Out[294]: 5

In [295]: next(rit)
Out[295]: 11

In [296]: next(rit)
Out[296]: 13

In [297]: next(rit)

    ---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)

    <ipython-input-297-b449dfcb3db0> in <module>()
----> 1 next(rit)


    StopIteration:

In [300]: r.stop
Out[300]: 15

In [302]: rit
Out[302]: <range_iterator object at 0x10589bc00>

In [303]: rit2
Out[303]: <range_iterator object at 0x10589b8d0>
```

0.5 Tuple Packing/Unpacking

```
In [304]: def getparts(string, sep=None):
              parts = string.split(sep)
              count = len(parts)
              return parts # but what if we also want/need to return count?

In [305]: entry = 'HP,300,42.55'

In [307]: getparts(entry, ',')

Out[307]: ['HP', '300', '42.55']

In [334]: def getparts(string, sep=None):
              parts = string.split(sep)
              count = len(parts)
              return parts, count, 'apple', 42 # tuple packing at work: creates a 2-tuple

In [323]: result = getparts(entry, ',')

In [324]: result

Out[324]: (['HP', '300', '42.55'], 3)

In [318]: stock = result[0]
          pcount = result[1]

In [319]: stock

Out[319]: ['HP', '300', '42.55']

In [320]: pcount

Out[320]: 3

In [321]: 'foo', 75, ['a', 'b', 'c']

Out[321]: ('foo', 75, ['a', 'b', 'c'])

In [325]: entry = 'AAPL,150,67.99'

In [330]: stock, count = getparts(entry, ',') # tuple unpacking

In [328]: stock

Out[328]: ['AAPL', '150', '67.99']

In [329]: count

Out[329]: 3

In [337]: stock, *remainder = getparts(entry, ',')

In [338]: stock

Out[338]: ['AAPL', '150', '67.99']

In [339]: remainder

Out[339]: [3, 'apple', 42]
```

```
In [340]: meta

Out[340]: ['foo', 'baz', 'zip', 'zap', 'blort', 'wibble', 'blort', 'wibble']

In [341]: enumerate(meta)

Out[341]: <enumerate object at 0x1074a3e58>

In [342]: list(enumerate(meta))

Out[342]: [(0, 'foo'), (1, 'baz'), (2, 'zip'), (3, 'zap'), (4, 'blort'), (5, 'wibble'), (6, 'blort'), (

In [343]: e = enumerate(meta)

In [344]: it = iter(e)

In [346]: next(it)

Out[346]: (0, 'foo')

In [347]: next(it)

Out[347]: (1, 'baz')

In [348]: next(it)

Out[348]: (2, 'zip')

In [349]: list(enumerate(meta, 22))

Out[349]: [(22, 'foo'), (23, 'baz'), (24, 'zip'), (25, 'zap'), (26, 'blort'), (27, 'wibble'), (28, 'blo

In [350]: for pair in enumerate(meta, 22):
              print(pair)

(22, 'foo')
(23, 'baz')
(24, 'zip')
(25, 'zap')
(26, 'blort')
(27, 'wibble')
(28, 'blort')
(29, 'wibble')

In [351]: for pair in enumerate(meta, 22):
              i = pair[0]
              w = pair[1]
              print('#', i, 'w:', w)

# 22 w: foo
# 23 w: baz
# 24 w: zip
# 25 w: zap
# 26 w: blort
# 27 w: wibble
# 28 w: blort
# 29 w: wibble
```

```
In [352]: for pair in enumerate(meta, 22):
              i, w = pair
              print('#', i, 'w:', w)

# 22 w: foo
# 23 w: baz
# 24 w: zip
# 25 w: zap
# 26 w: blort
# 27 w: wibble
# 28 w: blort
# 29 w: wibble

In [353]: for i, w in enumerate(meta, 22):
              print('#', i, 'w:', w)

# 22 w: foo
# 23 w: baz
# 24 w: zip
# 25 w: zap
# 26 w: blort
# 27 w: wibble
# 28 w: blort
# 29 w: wibble
```

0.6 Truthiness

Oh, what is `True`, and what is `False`?

In Python, **everything** is `True` (it **is** the *one true language* after all), **except**:

- `False`
- `None`
- `0, 0.0, 0+0j` (zero!)
- empty string
- empty containers
- instance with `inst.__bool__() == False` (`.__nonzero__()` in Python 2)

```
In [355]: things  = [3, 'foo', ('a', 'b'), dir, True]
          nothing = [0, False, 0.0, [], (), None]
          mix     = [3, 0, 'foo', '', None, [1,2,3]]

In [356]: # all() returns True if everything in the ITERABLE is Truthy, else False
          all(things)

Out[356]: True

In [357]: all(nothing)

Out[357]: False

In [358]: all(mix)

Out[358]: False

In [362]: # any() returns False if everything in the ITERABLE is Falsy, else True
          any(things)
```

```
Out[362]: True

In [360]: any(nothing)

Out[360]: False

In [361]: any(mix)

Out[361]: True

In [363]: class Point:
              def __init__(self, x, y):
                  self.x = x
                  self.y = y

              def __repr__(self):
                  return 'Point({x}, {y})'.format(x=self.x, y=self.y)

In [395]: class Curve:
              def __init__(self, points=None):
                  self.points = []
                  if points:
                      for p in points:
                          self.add(p)

              def add(self, p):
                  self.points.append(p)

              def __getitem__(self, index):
                  return self.points[index]

              def __len__(self):
                  return len(self.points)

              def __iter__(self):
                  return iter(self.points)

              def __repr__(self):
                  return 'Curve({pts})'.format(pts=repr(self.points))

              def __iadd__(self, other):
                  self.points.append(other)
                  return self

In [396]: a = Point(3, 5)
          b = Point(1, 2)
          c = Point(5, 7)
          curv = Curve()
          curv.add(a)
          curv.add(b)
          curv.add(c)

In [397]: a

Out[397]: Point(3, 5)

In [374]: curv
```

```
Out[374]: Curve([Point(3, 5), Point(1, 2), Point(5, 7)])

In [375]: curv.points

Out[375]: [Point(3, 5), Point(1, 2), Point(5, 7)]

In [376]: curv += Point(10, 12)

In [377]: curv

Out[377]: Curve([Point(3, 5), Point(1, 2), Point(5, 7), Point(10, 12)])

In [378]: curv += Point(15, 4)

In [379]: curv

Out[379]: Curve([Point(3, 5), Point(1, 2), Point(5, 7), Point(10, 12), Point(15, 4)])

In [380]: curv.add(Point(6,6))

In [381]: curv

Out[381]: Curve([Point(3, 5), Point(1, 2), Point(5, 7), Point(10, 12), Point(15, 4), Point(6, 6)])

In [382]: curv_clone = Curve([Point(3, 5), Point(1, 2), Point(5, 7), Point(10, 12), Point(15, 4), Point

In [383]: id(curv)

Out[383]: 4417417512

In [384]: id(curv_clone)

Out[384]: 4417208104

In [385]: # Can I iterate over my curve?
          # What does "Iteration on a Curve object mean?"
          for thing in curv:
              print(thing) # let's just see what we get!


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

    <ipython-input-385-850609ebfcaa> in <module>()
      1 # Can I iterate over my curve?
      2 # What does "Iteration on a Curve object mean?"
----> 3 for thing in curv:
      4     print(thing) # let's just see what we get!


    TypeError: 'Curve' object is not iterable


In [388]: curv

Out[388]: Curve([Point(3, 5), Point(1, 2), Point(5, 7)])

In [391]: curv[0] # because __getindex__
```

```
Out[391]: Point(3, 5)

In [393]: curv[2]

Out[393]: Point(5, 7)

In [394]: len(curv) # because __len__

Out[394]: 3

In [389]: for thing in curv:
              print(thing)

Point(3, 5)
Point(1, 2)
Point(5, 7)

In [398]: iter(curv)

Out[398]: <list_iterator object at 0x1074d9a90>

In [401]: i = dict(f='Ian', l='Stokes-Rees', age=39, zip=13210)
          e = dict(f='Emily', l='Stokes-Rees', age=32, zip=13210)
          b = dict(f='Ben', l='Smith', age=25, zip=2138)
          d = dict(f='Derek', l='Gambone', age=22, zip=2445)
          peeps = [i, e, b, d]

In [402]: peeps

Out[402]: [{'age': 39, 'l': 'Stokes-Rees', 'zip': 13210, 'f': 'Ian'}, {'age': 32, 'l': 'Stokes-Rees', 'z

In [403]: peeps.sort()


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

        <ipython-input-403-5ee159d7a672> in <module>()
    ----> 1 peeps.sort()


        TypeError: unorderable types: dict() < dict()


In [420]: def sortfunc(item):
              return (item['l'], item['f']) # sort by last name

In [421]: peeps.sort(key=sortfunc)

In [422]: peeps

Out[422]: [{'age': 22, 'l': 'Gambone', 'zip': 2445, 'f': 'Derek'},
           {'age': 25, 'l': 'Smith', 'zip': 2138, 'f': 'Ben'},
           {'age': 32, 'l': 'Stokes-Rees', 'zip': 13210, 'f': 'Emily'},
           {'age': 39, 'l': 'Stokes-Rees', 'zip': 13210, 'f': 'Ian'}]

In [423]: %pprint

Pretty printing has been turned OFF
```

0.7 lambda

It isn't Greek, it's Dutch for *Make Function*

`lambda` in Python allows us to create a single expression anonymous function

```
In [424]: def f(x):
              return 10 + x**2

In [425]: f(2)

Out[425]: 14

In [426]: f(3)

Out[426]: 19

In [427]: lambda x: 10 + x**2

Out[427]: <function <lambda> at 0x1074de9d8>

In [428]: f

Out[428]: <function f at 0x1074de488>

In [429]: g = lambda x: 7 + 2*x - 4*x**2

In [430]: g(2)

Out[430]: -5

In [431]: g(3)

Out[431]: -23

In [432]: g

Out[432]: <function <lambda> at 0x1074dea60>

In [434]: h = lambda x, y: 7 + 2*x + 3*x*y + y**2

In [435]: h(4, 8)

Out[435]: 175

In [436]: h = lambda x, y=2: 7 + 2*x + 3*x*y + y**2

In [437]: h(4, 8)

Out[437]: 175

In [438]: h(4)

Out[438]: 43

In [439]: peeps

Out[439]: [{'age': 22, 'l': 'Gambone', 'zip': 2445, 'f': 'Derek'}, {'age': 25, 'l': 'Smith', 'zip': 2138

In [440]: %pprint

Pretty printing has been turned ON
```

```
In [441]: peeps

Out[441]: [{'age': 22, 'l': 'Gambone', 'zip': 2445, 'f': 'Derek'},
           {'age': 25, 'l': 'Smith', 'zip': 2138, 'f': 'Ben'},
           {'age': 32, 'l': 'Stokes-Rees', 'zip': 13210, 'f': 'Emily'},
           {'age': 39, 'l': 'Stokes-Rees', 'zip': 13210, 'f': 'Ian'}]

In [442]: peeps.sort(key=lambda item: (item['l'], item['f']), reverse=True)

In [443]: peeps

Out[443]: [{'age': 39, 'l': 'Stokes-Rees', 'zip': 13210, 'f': 'Ian'},
           {'age': 32, 'l': 'Stokes-Rees', 'zip': 13210, 'f': 'Emily'},
           {'age': 25, 'l': 'Smith', 'zip': 2138, 'f': 'Ben'},
           {'age': 22, 'l': 'Gambone', 'zip': 2445, 'f': 'Derek'}]

In [444]: (lambda word: word.swapcase())('Ping Pong')

Out[444]: 'pING pONG'
```

0.8 Comprehensions

```
result = []
for VALS in ITERABLE:
    if COND(VALS):
        result.append(EXPR(VALS))
```

List comprehensions let us do this in a very clear and succinct way in a single expression (instead of multiple statements):

```
[ EXPR(VALS) for VALS in ITERABLE if COND(VALS) ]

In [446]: vals = [3, -8, 2, 7, 6, 2, 5, 12, 4, 9]
          words = 'foo bar ping pong blort wibble zip zap crunch'.split()

In [447]: evens = []
          for v in vals:
              if v%2 == 0:
                  evens.append(v)

In [448]: evens

Out[448]: [-8, 2, 6, 2, 12, 4]

In [461]: [v for v in vals if v%2 == 0]

Out[461]: [-8, 2, 6, 2, 12, 4]

In [449]: squares = []
          for v in vals:
              squares.append(v*v)

In [462]: squares

Out[462]: [9, 64, 4, 49, 36, 4, 25, 144, 16, 81]

In [463]: [v*v for v in vals]

Out[463]: [9, 64, 4, 49, 36, 4, 25, 144, 16, 81]
```

```
In [464]: bigsquares = []
          for v in vals:
              s = v*v
              if s > 10:
                  bigsquares.append(s)

In [455]: bigsquares

Out[455]: [64, 49, 36, 25, 144, 16, 81]

In [465]: [ v*v for v in vals if v*v > 10]

Out[465]: [64, 49, 36, 25, 144, 16, 81]

In [456]: capitals = []
          for w in words:
              capitals.append(w.upper())

In [457]: capitals

Out[457]: ['FOO', 'BAR', 'PING', 'PONG', 'BLORT', 'WIBBLE', 'ZIP', 'ZAP', 'CRUNCH']

In [466]: [ w.upper() for w in words ]

Out[466]: ['FOO', 'BAR', 'PING', 'PONG', 'BLORT', 'WIBBLE', 'ZIP', 'ZAP', 'CRUNCH']

In [458]: words

Out[458]: ['foo', 'bar', 'ping', 'pong', 'blort', 'wibble', 'zip', 'zap', 'crunch']

In [459]: longwords = []
          for w in words:
              if len(w) > 4:
                  longwords.append(w)

In [460]: longwords

Out[460]: ['blort', 'wibble', 'crunch']

In [468]: [ w for w in words if len(w) > 4 ]

Out[468]: ['blort', 'wibble', 'crunch']

In [469]: {w: len(w) for w in words}

Out[469]: {'bar': 3,
           'zip': 3,
           'pong': 4,
           'crunch': 6,
           'blort': 5,
           'zap': 3,
           'wibble': 6,
           'ping': 4,
           'foo': 3}

In [470]: colors = 'red RED pink blue GrEEn PINk BluE yelloW'.split()

In [471]: {c.lower() for c in colors}

Out[471]: {'blue', 'green', 'pink', 'red', 'yellow'}
```

0.9 Generator comprehensions

Often we are only interested in reading the results of our list comprehension once, and in order. If we DO NOT require random reads from the list, then we should use *Generator Comprehension* instead.

- read only
- read once
- read in order

Values are only calculated on-demand

```
In [473]: # identical to list comp but round brackets instead
          (v for v in vals if v%2 == 0)

Out[473]: <generator object <genexpr> at 0x1074e0708>

In [474]: g = (v for v in vals if v%2 == 0)

In [475]: g

Out[475]: <generator object <genexpr> at 0x1074e0ab0>

In [476]: next(g)

Out[476]: -8

In [477]: next(g)

Out[477]: 2

In [478]: next(g)

Out[478]: 6

In [479]: next(g)

Out[479]: 2

In [480]: next(g)

Out[480]: 12

In [481]: next(g)

Out[481]: 4
```

0.10 Counter

```
In [482]: !pwd

/Users/ijstokes/Dropbox/Public/python-mastery-isr19/python-mastery-isr19-code

In [483]: from collections import Counter

In [503]: with open('../student/Data/words.txt') as fh:
              words = fh.read().split()

In [487]: fh

Out[487]: <_io.TextIOWrapper name='../student/Data/words.txt' mode='r' encoding='UTF-8'>
```

```
In [488]: fh.seek(0)
```

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

<ipython-input-488-f1705fa1053c> in <module>()
----> 1 fh.seek(0)


ValueError: I/O operation on closed file.
```

```
In []:
```

```
In [504]: c = Counter(words)
```

```
In [486]: words
```

```
Out[486]: ['look',
           'into',
           'my',
           'eyes',
           'look',
           'into',
           'my',
           'eyes',
           'the',
           'eyes',
           'the',
           'eyes',
           'the',
           'eyes',
           'not',
           'around',
           'the',
           'eyes',
           "don't",
           'look',
           'around',
           'the',
           'eyes',
           'look',
           'into',
           'my',
           'eyes',
           "you're",
           'under']
```

```
In [505]: c
```

```
Out[505]: Counter({'eyes': 8, 'the': 5, 'look': 4, 'into': 3, 'my': 3, 'around': 2, 'not': 1, "you're":
```

```
In [506]: c.most_common(3)
```

```
Out[506]: [('eyes', 8), ('the', 5), ('look', 4)]
```

```
In [491]: c['eyes']

Out[491]: 8

In [492]: 'eyes' in c

Out[492]: True

In [507]: c.update('''
          eyes the eyes oh the eyes
          I don't totally understand where all this comes
          from, but I'm going with it
          '''.split())

In [508]: c

Out[508]: Counter({'eyes': 11, 'the': 7, 'look': 4, 'my': 3, 'into': 3, "don't": 2, 'around': 2, 'totall

In [496]: for w in ['foo', 'bar', 'zip', 'zap']:
              print(w)

foo
bar
zip
zap

In [501]: for w in 'foo bar zip zap'.split():
              print(w)

foo
bar
zip
zap

In [502]: [w.upper() for w in 'foo bar zip zap'.split()]

Out[502]: ['FOO', 'BAR', 'ZIP', 'ZAP']

In [509]: 'head' in c

Out[509]: False

In [510]: 'the' in c

Out[510]: True

In [511]: c['the']

Out[511]: 7
```

0.11 deque

Pronounced *DECK*, think *Card Deck*

Proposed and implemented by Raymond Hettinger, Pythonista Extraordinaire, and resident of the Valley.

Like a list, but good for:

- random inserts
- random deletes

- circular buffer
- fixed size list

At what cost?

- slightly slower index lookup
- slightly larger data structure

Goal: never to do a `memcpy`

```
In [512]: from collections import deque

In [513]: d = deque()

In [514]: d.extend([10, 20, 30, 40])

In [515]: d

Out[515]: deque([10, 20, 30, 40])

In [516]: d.pop()

Out[516]: 40

In [517]: d

Out[517]: deque([10, 20, 30])

In [518]: d.append(44)

In [519]: d[1]

Out[519]: 20

In [521]: d

Out[521]: deque([10, 20, 30, 44])

In [522]: d.popleft()

Out[522]: 10

In [523]: d

Out[523]: deque([20, 30, 44])

In [524]: d.appendleft(11)

In [525]: d

Out[525]: deque([11, 20, 30, 44])

In [526]: d.rotate(1)

In [527]: d

Out[527]: deque([44, 11, 20, 30])

In [528]: d.rotate(-1)

In [529]: d.rotate(-1)
```

```
In [530]: d
Out[530]: deque([20, 30, 44, 11])
In [531]: e = deque(maxlen=5)
In [532]: e.append(10)
In [533]: e.append(20)
In [534]: e.append(30)
In [535]: e
Out[535]: deque([10, 20, 30], maxlen=5)
In [536]: e.append(40)
In [537]: e.append(50)
In [538]: e.append(60)
In [539]: e
Out[539]: deque([20, 30, 40, 50, 60], maxlen=5)
In [540]: e.append(70)
In [541]: e
Out[541]: deque([30, 40, 50, 60, 70], maxlen=5)
In [542]: e.appendleft(22)
In [543]: e.appendleft(11)
In [544]: e
Out[544]: deque([11, 22, 30, 40, 50], maxlen=5)
```

0.12 Container Memory Consumption

```
In [545]: d = dict()
In [546]: import sys
In [547]: sys.getsizeof(d)
Out[547]: 288
In [548]: d['foo'] = 52
In [549]: sys.getsizeof(d)
Out[549]: 288
In [550]: d['bar'] = 21
In [551]: sys.getsizeof(d)
Out[551]: 288
```

```
In [552]: things = dict(zip=1, zap=2, ping=3, pong=4)

In [553]: d.update(things)

In [554]: d

Out[554]: {'bar': 21, 'pong': 4, 'zip': 1, 'zap': 2, 'ping': 3, 'foo': 52}

In [555]: things

Out[555]: {'pong': 4, 'ping': 3, 'zip': 1, 'zap': 2}

In [556]: sys.getsizeof(d)

Out[556]: 480

In [557]: d['blort'] = 5

In [559]: sys.getsizeof(d)

Out[559]: 480

In [560]: l = list()

In [561]: sys.getsizeof(l)

Out[561]: 64

In [562]: l.append(10)

In [563]: sys.getsizeof(l)

Out[563]: 96

In [564]: l.append(20)

In [565]: sys.getsizeof(l)

Out[565]: 96

In [566]: l.extend([30, 40])

In [567]: sys.getsizeof(l)

Out[567]: 96

In [568]: l.append(50)

In [569]: sys.getsizeof(l)

Out[569]: 128

In [571]: 128-96

Out[571]: 32

In [572]: 32 / 8

Out[572]: 4.0
```

```
In [574]: # NO!!!! Please don't ever do this in Python
          from random import choice

          # pick a word 7 times

          picks = 7
          while picks > 0:
              print(choice('red green yellow'.split()))
              picks -= 1
```

```
yellow
yellow
green
yellow
green
yellow
green
```

```
In [575]: # The Pythonic way to drive iteration is the for loop
          for pick in range(7):
              print(choice('red green yellow'.split()))
```

```
red
yellow
red
red
green
red
green
```

0.13 Lotto Challenge

- `lotto()` returns a random number from (1,100000)
- 10k tickets with random numbers are given out (may have duplicates)
- 1k "winning" numbers are drawn (may have duplicates)
- how many unique tickets win?
- how many people win?

```
In [577]: from random    import randint
          from functools import partial

          # some constants
          POOL = int(1e5) # size of pool we are selecting numbers from
          TCNT = int(1e4) # number of tickets we'll issue
          WCNT = int(1e3) # number of winning numbers will generate

          lotto = partial(randint, 1, POOL)
          # lotto = lambda : randint(1, POOL)

          tickets = [lotto() for draw in range(TCNT)] # create TCNT tickets
          uniqtix = set(tickets)                      # get unique set of tickets
          wintix  = {lotto() for draw in range(WCNT)} # get unique winners from WCNT draws
          # len(wintix) <= WCNT

          winnums = wintix & uniqtix                  # unique winning numbers
```

```
          winpeople = [ticket for ticket in tickets if ticket in winnums]

          print("Winning numbers:", len(winnums))
          print("Winning people:", len(winpeople))
```

```
Winning numbers: 104
Winning people: 107
```

```
In []: winnums.__contains__(ticket) # ticket in winnums
```

```
In [578]: class Point:
              pass
```

```
In [579]: a = Point()
          b = Point()
```

```
In [580]: hash(a)
```

```
Out[580]: -9223372036578525671
```

```
In [581]: hash(b)
```

```
Out[581]: 276250134
```

```
In [582]: a.__hash__()
```

```
Out[582]: -9223372036578525671
```

```
In [583]: b.__hash__()
```

```
Out[583]: 276250134
```

```
In [584]: d = dict(foo=1, bar=2)
```

```
In [585]: d
```

```
Out[585]: {'bar': 2, 'foo': 1}
```

```
In [586]: hash(d)
```

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

        <ipython-input-586-a37dc9dc2032> in <module>()
    ----> 1 hash(d)


        TypeError: unhashable type: 'dict'
```

```
In [593]: x = 55
          y = 56
```

```
In [588]: hash(x)
```

```
Out[588]: 55
```

```
In [589]: hash(y)

Out[589]: 56

In [590]: class Python3Integer:
              def __hash__(self):
                  return self%(2**64)

In [597]: hash('abcd')

Out[597]: -5124715813242141158

In [598]: hash('abce')

Out[598]: 8076609819467817677

In []:
```