01-classes2

January 27, 2017

```
In [39]: # Ignore the code in this cell!!

         import svgwrite
         import collections

         nobinding = "nobinding"

         def binding(var):
             try:
                 return eval(var)
             except NameError:
                 return nobinding

         class listis:
             def __init__(self):
                 self.lis = []
             def get(self, key):
                 for k,v in self.lis:
                     if key is k:
                         return v
             def put(self, key, val):
                 new = True
                 for pair in self.lis:
                     if pair[0] is key:
                         pair[1].append(val)
                         new = False
                 if new:
                     self.lis.append([key, [val]])
             def keys(self):
                 return [k for k,v in self.lis]

         class memgraph:
             def __init__(self, vars):
                 self.vars = sorted(vars)

             def _repr_svg_(self):
                 d = svgwrite.Drawing(size=(800,150))
```

```
        left = 100
        right = 260
        dy = 30
        vv = listis()
        ais = listis()

        for var in self.vars:
            val = binding(var)
            if val != nobinding:
                vv.put(val,var)
                ais.put(val, val)

        vals = ais.keys()
        vary = dict()

        y = dy
        d.add(d.text("Variables", insert=(left, y), text_anchor="end", fil
        y += dy

        for var in self.vars:
            d.add(d.text(var, insert=(left, y), text_anchor="end", fill='b
            vary[var] = y
            y += dy

        y = dy
        d.add(d.text("Objects(in the Heap)", insert=(right, y), fill='blue
        y += dy

        for val in vals:
            d.add(d.text(str(val), insert=(right, y), fill='black'))

            for var in vv.get(val):
                ly = vary[var]
                d.add(d.line((left, ly ), (right, y),  stroke=svgwrite.rgb
            y += dy

        return d.tostring()

    def svg(self):
        return self._repr_svg_()
```

1 Dictionary

- links objects in key/value pairs
- key order is undefined
- also known as a map, association, or hash table
- built into the language - another python workhorse

- type name is 'dict'

```
In [1]: #  two ways to make a empty dictionary

        [{}, dict(), type({})]

Out[1]: [{}, {}, dict]

In [2]: # dictionaries are written with curly '{}' brackets, and
        # key:value elements

        d = {'school':'columbia', 'class':'python', 'size':44}

In [3]: # len returns number of key/value pairs

        len(d)

Out[3]: 3

In [ ]: d['school']

In [4]: # add a key/value

        d['dept'] = 'comp sci'
        d

Out[4]: {'class': 'python', 'dept': 'comp sci', 'school': 'columbia', 'size': 44}

In [5]: # if you ask for a key that doesn't exist,
        # you'll get an error

        d['state']


        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-5-3e4806591922> in <module>()
          2 # you'll get an error
          3
    ----> 4 d['state']


        KeyError: 'state'


In [6]: # you can check for a key w/o an error
        # by using 'in'

        ['dept' in d, 'state' in d]
```

```
Out[6]: [True, False]

In [7]: # keys come back in an unpredictable order

        d.keys()

Out[7]: dict_keys(['dept', 'size', 'school', 'class'])

In [8]: # list of values

        d.values()

Out[8]: dict_values(['comp sci', 44, 'columbia', 'python'])

In [9]: # list of (k,v) tuples

        iview = d.items()
        iview

Out[9]: dict_items([('dept', 'comp sci'), ('size', 44), ('school', 'columbia'), ('c
```

1.1 Dictionary Views

- similiar to database view concept
- keys, values, items methods return 'live views', not 'dead lists'
- views always reflect the current contents of the dict

```
In [10]: ilist = list(d.items())
         ilist

Out[10]: [('dept', 'comp sci'),
          ('size', 44),
          ('school', 'columbia'),
          ('class', 'python')]

In [11]: # iv, the 'live' items view, has the new item

         d['new'] = 'thing'
         iview

Out[11]: dict_items([('dept', 'comp sci'), ('new', 'thing'), ('size', 44), ('schoo

In [12]: # the 'dead' list, does not

         ilist

Out[12]: [('dept', 'comp sci'),
          ('size', 44),
          ('school', 'columbia'),
          ('class', 'python')]
```

```
In [13]: # any object can be a value, but only immutable
         # objects can serve as keys
         # so, a list can't be a key

         d = dict()
         d[[1,2,3]] = "val"
```

```
        ----------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-13-8ce9af8b98bf> in <module>()
          4
          5 d = dict()
    ----> 6 d[[1,2,3]] = "val"


        TypeError: unhashable type: 'list'
```

```
In [14]: # but a tuple can be a key

         d = dict()
         d[(1,2,3)] = "val"
```

```
In [15]: # can make a dictionary with a
         # dictionary comprehension

         d = {x:x+10 for x in range(5)}
         d
```

```
Out[15]: {0: 10, 1: 11, 2: 12, 3: 13, 4: 14}
```

```
In [16]: # can use a comprehension to make a subset of a dictionary

         {k:d[k] for k in d if k <3}
```

```
Out[16]: {0: 10, 1: 11, 2: 12}
```

```
In [ ]: dir(dict)
```

2 Sets

- element order is undefined
- duplicates not allowed
- written with items inside '{}'
  - unlike dictionaries, no ':'

- type name is 'set'

```
In [17]: # 'set' expanded range
         # 'len' returns the number of elements in the set

         s1 = set(range(4,10))
         s2 = set(range(8, 12))
         [s1,s2, type(s1), len(s1)]

Out[17]: [{4, 5, 6, 7, 8, 9}, {8, 9, 10, 11}, set, 6]
```

```
In [18]: # note that the set constructor takes an "iterable"
         # something that produces a sequence of values
         # 34 is NOT an iterable so this bombs

         set(34)


        ---------------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-18-ebc74c361251> in <module>()
          3 # 34 is NOT an iterable so this bombs
          4
    ----> 5 set(34)


        TypeError: 'int' object is not iterable
```

```
In [19]: # to make a set with one element, do

         s = set()
         s.add(34)
         s

Out[19]: {34}
```

```
In [20]: # order doesn't matter

         {4,5,2} == {2,4,5}

Out[20]: True
```

```
In [21]: # duplicates are not allowed in a set

         {2,2,3,4,5,5,6}

Out[21]: {2, 3, 4, 5, 6}
```

```
In [22]: # intersection

         s1 & s2

Out[22]: {8, 9}

In [23]: # union - eliminates duplicates

         s1 | s2

Out[23]: {4, 5, 6, 7, 8, 9, 10, 11}

In [24]: # membership

         [7 in s1, 12 in s2]

Out[24]: [True, False]

In [25]: # set difference - elements in A but not in B

         s1 - s2

Out[25]: {4, 5, 6, 7}

In [26]: # elements in one set but not both

         s1 ^ s2

Out[26]: {4, 5, 6, 7, 10, 11}

In [27]: # add and remove set elements

         s1.add(33)
         s2.remove(9)
         [s1, s2]

Out[27]: [{4, 5, 6, 7, 8, 9, 33}, {8, 10, 11}]

In [28]: # can make a set with a
         # set comprehension

         {j*j for j in range(-4,10)}

Out[28]: {0, 1, 4, 9, 16, 25, 36, 49, 64, 81}
```

3 Example - anagrams

- words that use the same letters

```
In [29]: # a string iterable produces the chars in the string

         set('adsf')

Out[29]: {'a', 'd', 'f', 's'}

In [30]: def anagram(s1, s2):
             set1 = set(s1)
             set2 = set(s2)
             return set1 == set2

In [31]: # seems to work ok?

         [anagram('cat', 'dog'), anagram('silent', 'listen')]

Out[31]: [False, True]

In [32]: # well, not quite...

         anagram('a', 'aa')

Out[32]: True
```

4 Set methods

```
In [33]: dir(set)

Out[33]: ['__and__',
          '__class__',
          '__contains__',
          '__delattr__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__gt__',
          '__hash__',
          '__iand__',
          '__init__',
          '__ior__',
          '__isub__',
          '__iter__',
          '__ixor__',
          '__le__',
          '__len__',
          '__lt__',
          '__ne__',
          '__new__',
          '__or__',
          '__rand__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__ror__',
          '__rsub__',
          '__rxor__',
          '__setattr__',
          '__sizeof__',
          '__str__',
          '__sub__',
          '__subclasshook__',
          '__xor__',
          'add',
          'clear',
          'copy',
          'difference',
          'difference_update',
          'discard',
          'intersection',
          'intersection_update',
          'isdisjoint',
          'issubset',
          'issuperset',
          'pop',
          'remove',
          'symmetric_difference',
          'symmetric_difference_update',
          'union',
          'update']
```

5 Some objects can be ordered

- can do N-compares

```
In [34]: 3<7
```

```
Out[34]: True
```

```
In [35]: 3<6<5
```

```
Out[35]: False
```

```
In [36]: 3 < 5 < 8 < 9 < 11 < 13
```

```
Out[36]: True
```

```
In [37]: 'AAA' < 'AAX'

Out[37]: True
```

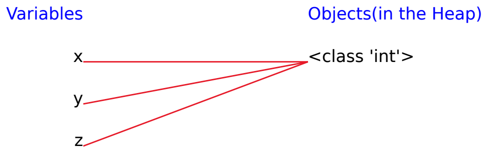
6 More about types

```
In [40]: # types are singletons

         x = type(234)
         y = type(2)
         z = int

         memgraph(['x','y','z'])

Out[40]:
```



```
In [41]: # type names are also class constructor functions
         # convert strings to ints and floats, and vv

         [int('345'), float('3.34'), str(234), str(3.4)]

Out[41]: [345, 3.34, '234', '3.4']

In [42]: # no arg usually produces a "default" value

         [int(), float(), str()]

Out[42]: [0, 0.0, '']

In [43]: # isinstance predicate
         # a little nicer than
         # type(34) == int

         [isinstance(34, int), isinstance(34, float)]

Out[43]: [True, False]

In [44]: # can test for several types at once

         [isinstance(34, (int, float)),
          isinstance(234.234, (int,float)),
          isinstance('asdf', (int,float))]

Out[44]: [True, True, False]
```

7 Objects vs String Representation of an Object

- The ‘string representation’ is derived from an object, but should not be confused with the object itself.
- A given object can have multiple string Representations
  - two different strings might refer to the same object
    - ‘larry’ vs ‘larry stead’
  - two identical strings might refer to different objects
    - ‘larry’ and ‘larry’. first ‘larry’ might refer to ‘larry stead’, the second to ‘larry smith’
- also, some tools and versions of Python may print things slightly differently
  - ipython pretty printer - attempts to print complex objects in a form readable by humans
- we will see how this works in detail later

```
In [45]: # example - int
         # we see the same int object printed two different ways below

         print(int)

<class 'int'>
```

```
In [46]: # 'str' function converts object into a string representation

         str(int)

Out[46]: "<class 'int'>"
```

```
In [47]: # but here int prints differently
         # why?

         int

Out[47]: int
```

```
In [48]: # it turns out ipython has a 'pretty printer' - which has its own notion
         # of 'what looks nice'
         # let's turn it OFF - then we get the same string as above
         # some people think 'int' is prettier than '<class 'int'>

         %pprint

         int

Pretty printing has been turned OFF


Out[48]: <class 'int'>
```

```
In [49]: # note however, that string reps are NOT always valid input

         <class 'int'>


           File "<ipython-input-49-85327f57862d>", line 3
         <class 'int'>
         ^
     SyntaxError: invalid syntax
```

```
In [50]: # another example
         # pretty printer is still off

         list(range(50))

Out[50]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
```

```
In [51]: # turn it back on

         %pprint

         # now we get a one item per line print out, which could be helpful for 'wi
         # but doesn't seem useful for small integers. (the pretty printer could be
         # little smarter about this)

         list(range(50))

Pretty printing has been turned ON


Out[51]: [0,
          1,
          2,
          3,
          4,
          5,
          6,
          7,
          8,
          9,
          10,
          11,
          12,
          13,
          14,
          15,
          16,
 17,
 18,
 19,
 20,
 21,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31,
 32,
 33,
 34,
 35,
 36,
 37,
 38,
 39,
 40,
 41,
 42,
 43,
 44,
 45,
 46,
 47,
 48,
 49]
```

```
In [52]: # for these big ints, the pretty printer looks better...

         [2**n for n in range(1000, 1004)]
```

```
Out[52]: [1071508607186267320948425049060001810561404811705533607443750388370351051
          2143017214372534641896850098120003621122809623411067214887500776740702102
          4286034428745069283793700196240007242245619246822134429775001553481404204
          8572068857490138567587400392480014484491238493644268859550003106962808408
```

```
In [53]: # than this...

         %pprint

         [2**n for n in range(1000, 1004)]
```

```
Pretty printing has been turned OFF
```

```
Out[53]: [1071508607186267320948425049060001810561404811705533607443750388370351051
```

```
In [ ]:
```