515_03_02_Data_Structures

November 14, 2024

1 Data Structures

1.1 Lists

[1]:
```
empty_list = list()
empty_list = []
```

[2]:
```
len(empty_list)
```

[2]:
```
0
```

[4]:
```
empty_list[0]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[4], line 1
----> 1 empty_list[0]

IndexError: list index out of range
```

[5]:
```
squares = [1, 4 ,9 ,25, 36]
```

[6]:
```
squares
```

[6]:
```
[1, 4, 9, 25, 36]
```

[7]:
```
squares[0]
```

[7]:
```
1
```

[8]:
```
squares[-1]
```

[8]:
```
36
```

[9]:
```
squares[1:3]
```

[9]:
```
[4, 9]
```

[10]:
```
squares[:2]
```

[10]:
```
[1, 4]
```

[11]:
```
squares[2:]
```

[11]:
```
[9, 25, 36]
```

[12]:
```
squares[:]
```

[12]:
```
[1, 4, 9, 25, 36]
```

1.1.1 Lists of lists

[13]:
```
cubes = [1, 8, 27, 64, 125]
```

[14]:
```
powers = [squares, cubes]
```

[15]:
```
powers
```

[15]:
```
[[1, 4, 9, 25, 36], [1, 8, 27, 64, 125]]
```

[16]:
```
powers[0][2]
```

[16]:
```
9
```

1.1.2 Shallow vs. deep copying

[17]:
```
copy_of_squares = squares
```

[18]:
```
copy_of_squares.append(49)
```

[19]:
```
copy_of_squares
```

[19]:
```
[1, 4, 9, 25, 36, 49]
```

[20]:
```
squares
```

[20]:
```
[1, 4, 9, 25, 36, 49]
```

[21]:
```
deep_copy_of_squares = squares.copy()
```

[22]:
```
deep_copy_of_squares
```

[22]:
```
[1, 4, 9, 25, 36, 49]
```

[23]:
```
deep_copy_of_squares.append(64)
```

[24]:
```
deep_copy_of_squares
```

[24]:
```
[1, 4, 9, 25, 36, 49, 64]
```

[25]:
```
squares
```

[25]:
```
[1, 4, 9, 25, 36, 49]
```

1.1.3 Append vs. extend

[26]:
```
squares.append([64, 81])
```

[27]:
```
squares
```

[27]:
```
[1, 4, 9, 25, 36, 49, [64, 81]]
```

[28]:
```
del squares[-1]
```

[29]:
```
squares.extend([64, 81])
```

[30]:
```
squares
```

[30]:
```
[1, 4, 9, 25, 36, 49, 64, 81]
```

1.1.4 List comprehensions

[31]:
```
squares = []
for x in range(10):
    squares.append(x**2)
```

[32]:
```
squares = list(map(lambda x: x**2, range(10)))
```

[33]:
```
squares = [x**2 for x in range(10)]
```

1.2 Tuples

Similar to lists, but immutable [WHY IS THIS IMPORTANT?]

[34]:
```
latitude = 42.314081
longitude = 83.036857
```

1.2.1 Tuple packing

[35]:
```
coord = (latitude, longitude)
```

[36]:
```
coord
```

[36]:
```
(42.314081, 83.036857)
```

1.2.2 Sequence unpacking

[37]:
```
coord_lat, coord_lon = coord
```

1.2.3 Special cases: 1-tuples and 0-tuples

[38]:
```
empty_tuple = ()
```

[39]:
```
empty_tuple
```

[39]:
```
()
```

[40]:
```
one_tuple = "mads",  # note trailing comma
```

[41]:
```
one_tuple
```

[41]:
```
('mads',)
```

1.3 Sets

A set is an unordered collection with no duplicate elements.

[42]:
```
animals = {'dog', 'cat', 'horse', 'dog'}
```

[43]:
```
animals
```

[43]:
```
{'cat', 'dog', 'horse'}
```

[44]:
```
'dog' in animals
```

[44]:
```
True
```

[45]:
```
'turtle' in animals
```

[45]:
```
False
```

[46]:
```
letters = set('antidisestablishmentarianism') # Longest non-contrived and␣
↪nontechnical English word
```

[47]:
```
letters
```

[47]:
```
{'a', 'b', 'd', 'e', 'h', 'i', 'l', 'm', 'n', 'r', 's', 't'}
```

[48]:
```
len(letters)
```

[48]:
```
12
```

1.4 Dictionaries

Indexed by keys, which are any immutable data type (i.e. not lists or sets). Tuples are legitimate keys as long as they don't contain any mutable types. Strings are commonly used as keys.

[49]:
```
city_population = {'Chongqing': 30165500,
           'Shanghai': 24183300,
           'Beijing': 21707000,
           'Istanbul': 15029231,
           'Karachi': 14910352,
           'Dhaka': 14399000,
           'Tokyo': 13515271,
           'Moscow': 13200000}
```

[50]:
```
city_population
```

[50]:
```
{'Chongqing': 30165500,
 'Shanghai': 24183300,
 'Beijing': 21707000,
 'Istanbul': 15029231,
 'Karachi': 14910352,
 'Dhaka': 14399000,
 'Tokyo': 13515271,
 'Moscow': 13200000}
```

[51]:
```
city_population['Dhaka']
```

[51]:
```
14399000
```

[52]:
```
'Beijing' in city_population
```

[52]:
```
True
```

[53]:
```
'Ann Arbor' in city_population
```

[53]:
```
False
```

[54]:
```
city_population['Ann Arbor'] = 113934
```

[55]:
```
'Ann Arbor' in city_population
```

[55]:
```
True
```

[56]:
```
for city, population in city_population.items():
    print('The population of {0} is {1}.'.format(city, population))
```

```
The population of Chongqing is 30165500.
The population of Shanghai is 24183300.
The population of Beijing is 21707000.
The population of Istanbul is 15029231.
The population of Karachi is 14910352.
The population of Dhaka is 14399000.
The population of Tokyo is 13515271.
The population of Moscow is 13200000.
The population of Ann Arbor is 113934.
```

1.5 collections

- Counter
- defaultdict
- deque
- namedtuple

1.5.1 Counter

[57]:
```
from collections import Counter
```

[58]:
```
c_empty = Counter()                                # a new, empty counter
c_longest_word = Counter('antidisestablishmentarianism')                 # a␣
 ↪new counter from an iterable
c_colors = Counter({'red': 4, 'blue': 2})      # a new counter from a mapping
c_animals = Counter(cats=4, dogs=8)
```

[59]:
```
c_longest_word
```

[59]:
```
Counter({'i': 5,
         'a': 4,
         's': 4,
         'n': 3,
         't': 3,
         'e': 2,
         'm': 2,
         'd': 1,
         'b': 1,
         'l': 1,
         'h': 1,
         'r': 1})
```

[60]:
```
c_longest_word.most_common(1)
```

[60]:
```
[('i', 5)]
```

[61]:
```
c_longest_word.popitem()
```

[61]:
```
('r', 1)
```

[62]:
```
c_longest_word
```

[62]:
```
Counter({'i': 5,
         'a': 4,
         's': 4,
         'n': 3,
         't': 3,
         'e': 2,
         'm': 2,
         'd': 1,
         'b': 1,
         'l': 1,
         'h': 1})
```

[63]:
```
counts = {'u':1,'i':4,'u':8}
```

[64]:
```
Counter(counts).most_common(1)
```

[64]:
```
[('u', 8)]
```

[65]:
```
things_to_count=['a','a','a','a','b','b','b','b','b']
```

[66]:
```
Counter(things_to_count).most_common(1)
```

[66]:
```
[('b', 5)]
```

[ ]:
```

```

1.5.2 defaultdict

[67]:
```
fruit_totals = {}
fruit_counts = [('apple', 3),('pear', 2), ('orange', 1), ('apple', 1)]

for fruit,count in fruit_counts:
    if fruit in fruit_totals:
        fruit_totals[fruit] = fruit_totals[fruit] + count
    else:
        fruit_totals[fruit] = count
print(fruit_totals)
```

```
{'apple': 4, 'pear': 2, 'orange': 1}
```

[68]:
```
from collections import defaultdict
fruit_totals = defaultdict(int)
for fruit,count in fruit_counts:
    fruit_totals[fruit] = fruit_totals[fruit] + count
print(fruit_totals)
```

```
defaultdict(<class 'int'>, {'apple': 4, 'pear': 2, 'orange': 1})
```

[69]:
```
Counter(fruit_totals).most_common(1)
```

[69]:
```
[('apple', 4)]
```

[70]:
```
from collections import defaultdict
fruit_totals = defaultdict(list)
for fruit,count in fruit_counts:
    fruit_totals[fruit].append(count)
print(fruit_totals)
```

```
defaultdict(<class 'list'>, {'apple': [3, 1], 'pear': [2], 'orange': [1]})
```

1.5.3 deque

[71]:
```
from collections import deque
import itertools # beyond the scope of this course --
                 # we are using itertools.islice() to return a slice of an␣
 ↪iterator,
                 # which is like a slice of a list but more memory-efficient
```

[72]:
```
def moving_average(iterable, n=3):
    # moving_average([40, 30, 50, 46, 39, 44]) --> 40.0 42.0 45.0 43.0
    # http://en.wikipedia.org/wiki/Moving_average
    it = iter(iterable)
    d = deque(itertools.islice(it, n-1))
    d.appendleft(0)
    s = sum(d)
    for elem in it:
        s += elem - d.popleft()
        d.append(elem)
        yield s / n
```

[73]:
```
for i in moving_average([40, 30, 50, 46, 39, 44]):
    print(i)
```

```
40.0
42.0
```

```
45.0
43.0
```

1.5.4 namedtuple

[74]:
```
from collections import namedtuple
```

[75]:
```
Coord = namedtuple('Coord', ['latitude','longitude'])
```

[76]:
```
c = Coord(latitude = 42.314081, longitude = 83.036857)
```

[77]:
```
c[1]
```

[77]:
```
83.036857
```

[78]:
```
lat, lon = c
```

[79]:
```
c.latitude
```

[79]:
```
42.314081
```

[80]:
```
c
```

[80]:
```
Coord(latitude=42.314081, longitude=83.036857)
```

[81]:
```
d = {'latitude': 42.314081, 'longitude': '83.036857'}
```

[82]:
```
c = Coord(**d)
```

[83]:
```
c
```

[83]:
```
Coord(latitude=42.314081, longitude='83.036857')
```

[ ]:
```

```