In [1]:
```
#data structure list
#my_list = []
my_list = list()
```

In [2]:
```
my_list
```

Out[2]:
```
[]
```

In [3]:
```
my_list.append(1)
```

In [4]:
```
my_list
```

Out[4]:
```
[1]
```

In [5]:
```
my_list.append(1)
```

In [6]:
```
my_list
```

Out[6]:
```
[1, 1]
```

In [7]:
```
my_list.append(5)
```

In [8]:
```
my_list
```

Out[8]:
```
[1, 1, 5]
```

In [9]:
```
my_list[2]
```

Out[9]:
```
5
```

In [10]:
```
my_list[:2]
```

Out[10]:
```
[1, 1]
```

In [11]:
```
my_list[1:]
```

Out[11]:
```
[1, 5]
```

In [12]:
```
my_list.append(634)
```

In [13]:
```
my_list
```

Out[13]:
```
[1, 1, 5, 634]
```

In [14]:
```
#indexing in PYTHON starts at position 0
```

In [15]:
```
#reverse order of a list
my_list[::-1]
```

Out[15]:
```
[634, 5, 1, 1]
```

In [16]:
```
my_copy_list = my_list
```

In [17]:
```
my_copy_list
```

Out[17]:
```
[1, 1, 5, 634]
```

In [18]:
```
my_list.append(77)
```

In [19]:
```
my_list
```

Out[19]:
```
[1, 1, 5, 634, 77]
```

In [20]:
```
my_copy_list
```

Out[20]:
```
[1, 1, 5, 634, 77]
```

In [21]:
```
#true copy of a list
proper_copy = my_copy_list.copy()
```

In [22]:
```
proper_copy
```

Out[22]:
```
[1, 1, 5, 634, 77]
```

In [23]:
```
my_list.append(434)
```

In [24]:
```
my_list
```

Out[24]:
```
[1, 1, 5, 634, 77, 434]
```

In [25]:
```
proper_copy
```

Out[25]:
```
[1, 1, 5, 634, 77]
```

In [26]:
```
new_list = [32, "columbia", 3.14, False]
```

In [27]:
```
new_list
```

Out[27]:
```
[32, 'columbia', 3.14, False]
```

In [28]:
```
proper_copy
```

Out[28]:
```
[1, 1, 5, 634, 77]
```

In [29]:
```
proper_copy.count(1)
```

Out[29]:
```
2
```

In [30]:
```
#set
#my_set = {}
my_set = set()
```

In [31]:
```
my_set
```

Out[31]:
```
set()
```

In [32]:
```
my_set.add("new york")
my_set.add("new york")
my_set.add("new jersey")
my_set.add("alaska")
```

In [33]:
```
my_set
```

Out[33]:
```
{'alaska', 'new jersey', 'new york'}
```

In [34]:
```
#sets honor distinct/dedup/unique
```

In [35]:
```
my_set.add("New York")
```

In [36]:
```
my_set
```

Out[36]:
```
{'New York', 'alaska', 'new jersey', 'new york'}
```

In [37]:
```
my_set_a = {"new york", "20-25", "audi", "fly fish"}
my_set_b = {"new jersey", "20-25", "bmw", "swim"}
```

In [38]:
```
#union - what attributes do ALL the sets have together
```

```
union_fun = my_set_a.union(my_set_b) #order does not matter
```

In [39]:
```
union_fun
```

Out[39]:
```
{'20-25', 'audi', 'bmw', 'fly fish', 'new jersey', 'new york', 'swim'}
```

In [40]:
```
#intersection - what do two sets have in common
intersection_fun = my_set_b) #order does NOT mattert_a.intersection(my_se
```

In [41]:
```
intersection_fun
```

Out[41]:
```
{'20-25'}
```

In [42]:
```
difference_fun = my_set_b.difference(my_set_a)
#order DOES matter my_set_b.difference(my_set_a) != my_set_a.difference(my_set_b
```

In [43]:
```
difference_fun
```

Out[43]:
```
{'bmw', 'new jersey', 'swim'}
```

In [44]:
```
symmetric_fun = my_set_b.symmetric_difference(my_set_a) #order does NOT matter
```

In [45]:
```
symmetric_fun
```

Out[45]:
```
{'audi', 'bmw', 'fly fish', 'new jersey', 'new york', 'swim'}
```

In [46]:
```
#dictionary
#key, value pair
dictionary_fun = dict()
```

In [47]:
```
dictionary_fun
```

Out[47]:
```
{}
```

In [48]:
```
dictionary_fun["key_a"] = "columbia"
dictionary_fun["key_b"] = "princeton"
```

In [49]:
```
dictionary_fun
```

Out[49]:
```
{'key_a': 'columbia', 'key_b': 'princeton'}
```

In [50]:
```
dictionary_fun["key_a"] = "harvard"
```

In [51]:
```
dictionary_fun
```

Out[51]:
```
{'key_a': 'harvard', 'key_b': 'princeton'}
```

In [52]:
```
dictionary_fun["key_c"] = "columbia"
```

In [53]:
```
dictionary_fun
```

Out[53]:
```
{'key_a': 'harvard', 'key_b': 'princeton', 'key_c': 'columbia'}
```

In [54]:
```
dictionary_inferred = {'key_a': 'harvard', 'key_b': 'princeton', 'key_c': 'colum
```

In [55]:
```
dictionary_inferred
```

Out[55]:
```
{'key_a': 'harvard', 'key_b': 'princeton', 'key_c': 'columbia'}
```

In [56]:
```
dictionary_fun_a = {"key_a": [1,5,6], "key_b": {"a": 1, "b": 2}}
```

In [57]:
```
dictionary_fun_a
```

Out[57]:
```
{'key_a': [1, 5, 6], 'key_b': {'a': 1, 'b': 2}}
```

In [58]:
```
dictionary_fun_a["key_b"]
```

Out[58]:
```
{'a': 1, 'b': 2}
```

In [59]:
```
type(dictionary_fun_a["key_a"])
```

Out[59]:
```
list
```

In [60]:
```
what_are_the_keys = list(dictionary_fun_a.keys())
```

In [61]:
```
what_are_the_keys
```

Out[61]:
```
['key_a', 'key_b']
```

In [62]:
```
what_are_the_values = list(dictionary_fun_a.values())
```

In [63]:
```
what_are_the_values
```

Out[63]:
```
[[1, 5, 6], {'a': 1, 'b': 2}]
```

In [64]:
```
#looping statements
list(range(1, 11))
#for i in range(0, 10):
```

Out[64]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [65]:
```
for whateveryouwant in range(1, 11):
    print (whateveryouwant)
```

```
1
2
3
4
5
6
7
8
9
10
```

In [66]:
```
cnt = 0
while cnt < 10:
    #cnt = cnt + 1
    cnt += 1
    print (cnt)
```

```
1
2
3
4
5
6
7
8
9
10
```

In [67]:
```
my_list_fun = ["a", "b", "c", "d", "e"]
```

In [68]:
```
concat_fun = list()
for djfkj in my_list_fun:
    concat_fun.append(djfkj + " fun")
    print (djfkj)
```

```
a
b
c
d
e
```

In [69]:
```
concat_fun
```

Out[69]:
```
['a fun', 'b fun', 'c fun', 'd fun', 'e fun']
```

In [70]:
```
sentence_fun = "the,cat,ran,up,the,hill,after,dog,who,beat,up,my,friends,cat"
```

In [71]:
```
sentence_fun
```

Out[71]:
```
'the,cat,ran,up,the,hill,after,dog,who,beat,up,my,friends,cat'
```

In [72]:
```
#tokenization
my_token_fun = sentence_fun.split(",") #defaults to a space
```

In [73]:
```
my_token_fun
```

Out[73]:
```
['the',
 'cat',
 'ran',
 'up',
 'the',
 'hill',
 'after',
 'dog',
 'who',
 'beat',
 'up',
 'my',
 'friends',
 'cat']
```

In [74]:
```
#what you all to create a dictionary where the keys are "UNIQUE" tokens
#and the values are the COUNT of the number of times that token showed up in the
```

In [75]:
```
#my_token_fun
word_freq = dict()
my_set = set(my_token_fun)
for word in my_set:
    word_freq[word] = my_token_fun.count(word)
```

In [76]:
```
word_freq
```

Out[76]:
```
{'the': 2,
 'hill': 1,
 'ran': 1,
 'friends': 1,
 'after': 1,
 'beat': 1,
 'my': 1,
 'up': 2,
 'cat': 2,
 'who': 1,
 'dog': 1}
```

In [77]:
```
#create a function whose input is a corpus and the function needs to
#output a word frequency dictionary
def word_count_fun(str_in):
```

```
    dict_fun = dict()
    token_tmp = str_in.split()
    for w in set(token_tmp):
        dict_fun[w] = token_tmp.count(w)
    return dict_fun
```

In [78]:
```
test = word_count_fun("the the cat dog dog")
```

In [79]:
```
test
```

Out[79]:
```
{'the': 2, 'dog': 2, 'cat': 1}
```

In [80]:
```
import collections
```

In [81]:
```
test_sent = "the the cat dog dog"
test_fun = collections.Counter(test_sent.split())
```

In [82]:
```
test_fun
```

Out[82]:
```
Counter({'the': 2, 'cat': 1, 'dog': 2})
```

In [83]:
```
#list comprehension
sentence_blah = "a b c d d e e e g g i"
```

In [84]:
```
#word_freq_lc = [word+" fun" for word in sentence_blah.split()]
word_freq_lc = list()
for word in sentence_blah.split():
    word_freq_lc.append(word + " fun")
```

In [85]:
```
word_freq_lc
```

Out[85]:
```
['a fun',
 'b fun',
 'c fun',
 'd fun',
 'd fun',
 'e fun',
 'e fun',
 'e fun',
 'g fun',
 'g fun',
 'i fun']
```

In [86]:
```
sentence_blah = "a b c d d e e e g g i"
dict_fun = {word:sentence_blah.count(word) for word in set(sentence_blah.split()
```

In [87]:
```
dict_fun
```

Out[87]:
```
{'c': 1, 'g': 2, 'b': 1, 'e': 3, 'i': 1, 'd': 2, 'a': 1}
```

In [88]:
```
import pandas as pd
```

In [89]:
```
my_pd = pd.DataFrame()
```

In [90]:
```
type(my_pd)
```

Out[90]:
```
pandas.core.frame.DataFrame
```

In [91]:
```
my_pd = pd.DataFrame()
for word in dict_fun.keys():
    my_pd = my_pd.append({"token": word, "freq": dict_fun[word]}, ignore_index=T
```

In [92]:
```
my_pd
```

Out[92]:

| | token | freq |
|---|---|---|
| **0** | c | 1.0 |
| **1** | g | 2.0 |
| **2** | b | 1.0 |
| **3** | e | 3.0 |
| **4** | i | 1.0 |
| **5** | d | 2.0 |
| **6** | a | 1.0 |

In [93]:
```
my_pd.head(1)
```

Out[93]:

| | token | freq |
|---|---|---|
| **0** | c | 1.0 |

In [94]:
```
my_pd.tail(1)
```

Out[94]:

| | token | freq |
|---|---|---|
| **6** | a | 1.0 |

In [95]:
```
stats = my_pd.describe()
```

In [96]:
```
stats
```

Out[96]:

| | freq |
|---|---|
| **count** | 7.000000 |
| **mean** | 1.571429 |
| **std** | 0.786796 |
| **min** | 1.000000 |
| **25%** | 1.000000 |
| **50%** | 1.000000 |
| **75%** | 2.000000 |
| **max** | 3.000000 |

In [97]:

```
my_pd
```

Out[97]:

| | token | freq |
|---|---|---|
| **0** | c | 1.0 |
| **1** | g | 2.0 |
| **2** | b | 1.0 |
| **3** | e | 3.0 |
| **4** | i | 1.0 |
| **5** | d | 2.0 |
| **6** | a | 1.0 |

In [98]:

```
my_pd["freq"]
```

Out[98]:

```
0    1.0
1    2.0
2    1.0
3    3.0
4    1.0
5    2.0
6    1.0
Name: freq, dtype: float64
```

In [99]:

```
my_pd.freq
```

Out[99]:

```
0    1.0
1    2.0
2    1.0
3    3.0
4    1.0
5    2.0
6    1.0
Name: freq, dtype: float64
```

In [100…

```
what_are_the_columns = my_pd.columns
```

In [101…
```
what_are_the_columns
```

Out[101…
```
Index(['token', 'freq'], dtype='object')
```

In [102…
```
my_pd.columns = ["freq_fun", "token"] #change names
```

In [103…
```
my_pd
```

Out[103…

| | freq_fun | token |
|---|---|---|
| **0** | c | 1.0 |
| **1** | g | 2.0 |
| **2** | b | 1.0 |
| **3** | e | 3.0 |
| **4** | i | 1.0 |
| **5** | d | 2.0 |
| **6** | a | 1.0 |

In [104…
```
my_pd_ex = my_pd[my_pd.freq_fun >= 2.0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/h_/ktb589cd4qd4t7wbtqgxcgnw0000gn/T/ipykernel_11647/112839197.py in
<module>
----> 1 my_pd_ex = my_pd[my_pd.freq_fun >= 2.0]

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/ops/common.py in new_met
hod(self, other)
     67         other = item_from_zerodim(other)
     68
---> 69         return method(self, other)
     70
     71     return new_method

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/arraylike.py in __ge__(s
elf, other)
     50     @unpack_zerodim_and_defer("__ge__")
     51     def __ge__(self, other):
---> 52         return self._cmp_method(other, operator.ge)
     53
     54     # -------------------------------------------------------------

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/series.py in _cmp_method
(self, other, op)
   5500
   5501         with np.errstate(all="ignore"):
-> 5502             res_values = ops.comparison_op(lvalues, rvalues, op)
   5503
   5504         return self._construct_result(res_values, name=res_name)
```

```
~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/ops/array_ops.py in comp
arison_op(left, right, op)
    282
    283     elif is_object_dtype(lvalues.dtype) or isinstance(rvalues, str):
--> 284         res_values = comp_method_OBJECT_ARRAY(op, lvalues, rvalues)
    285
    286     else:

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/ops/array_ops.py in comp
_method_OBJECT_ARRAY(op, x, y)
     71         result = libops.vec_compare(x.ravel(), y.ravel(), op)
     72     else:
---> 73         result = libops.scalar_compare(x.ravel(), y, op)
     74     return result.reshape(x.shape)
     75

~/opt/anaconda3/lib/python3.9/site-packages/pandas/_libs/ops.pyx in pandas._lib
s.ops.scalar_compare()

TypeError: '>=' not supported between instances of 'str' and 'float'
```

In [ ]:

```
my_pd
```

In [ ]:

```
my_pd_ex = my_pd[(my_pd.freq_fun == "a") or (my_pd.freq_fun == "a")]
```