### Set Functions (A fast-searchable set of unique values)

|  |  |
| --- | --- |
| add(value) | adds the given value to the set |
| discard(value)  | removes the given value from the set |
| clear() | removes all elements of the set |
| pop() | removes and returns a random element from the set |

Other useful operations (a and b are both sets):

|  |  |
| --- | --- |
| a - b | returns a new set containing values in a but not in b |
| a | b | returns a new set containing values in either a or b |
| a & b | returns a new set containing values in both a and b |
| a ^ b | returns a new set containing values in a or b but not both |

### Dictionary Functions (A fast mapping between a set of keys and a set of values)

|  |  |
| --- | --- |
| my\_dict[key] = value | adds a mapping from the given key to the given value |
| my\_dict[key] | returns the value mapped to the given key (error if not found) |
| popitem() | removes and returns an arbitrary (key, value) pair (error if empty) |
| pop(key) | removes any existing mapping for the given key (error if not found) |
| clear() | removes all key/value pairs from the dictionary |

Other useful functions:

|  |  |
| --- | --- |
| keys() | returns a Set of all keys in the map |
| values() | returns a Collection of all values in the map |
| items() | return a new view of the dictionary’s items ((key, value) pairs) |

###

# Problems

### Set<E>

1. Write a function num\_unique that takes a list of integers as a parameter and returns the number of unique integer values in the list. Use a set as auxiliary storage to help you solve this problem.

 For example, if a list contains the values [3, 7, 3, -1, 2, 3, 7, 2, 15, 15], your function should return 5. The empty list contains 0 unique values.

2. Write a function num\_in\_common that takes two lists of integers as parameters and returns the number of unique integers that occur in both lists. Use one or more sets as storage to help you solve this problem.

 For example, if one list contains the values [3, 7, 3, -1, 2, 3, 7, 2, 15, 15] and the other list contains the values [-5, 15, 2, -1, 7, 15, 36], your function should return 4 (because the elements -1, 2, 7, and 15 occur in both lists).

3. Write a function max\_length that takes a set of strings as a parameter and that returns the length of the longest string in the list. If your function is passed an empty set, it should return 0.

4. Write a function has\_odd that takes a set of integers as a parameter and that returns True if the set contains at least one odd integer, and False otherwise. If passed the empty set, your function should return False.

### Dictionary

5. Write a function contains3 that accepts a list of strings as a parameter and returns true if any single string occurs at least 3 times in the list, and False otherwise. Use a dictionary as auxiliary storage.

6. Write a function counts that accepts a list of integers and a set of integers as parameters, and returns a dictionary from each value in the set to the number of occurrences of that value in the list. For example, if your function is passed the following list and set as parameters:

list: [**4**, **-2**, 3, 9, **4**, 17, 5, **29**, 14, 87, **4**, **-2**, 100]

set: [-2, 4, 29]

 Then your function should return the dictionary {-2:2, 4:3, 29:1}, because there are two occurrences of -2, three occurrences of 4, and one occurrence of 29. The order of the key/value pairs does not matter.

# Problems (continued)

7. Write a function is\_unique that accepts a dictionary from strings to strings as a parameter and returns True if no two keys map to the same value (and False if any two or more keys do map to the same value). For example, calling your function on the following map would return True:

{Allison=Obourn, Lester=McCann, Patrick=Homer, Rick=Mercer, Bill=Mitchell}

 Calling it on the following dictionary would return False, because of two mappings for Homer and Mercer:

{Kendrick=Mercer, Lester=McCann, Patrick=Homer, Robin=Homer, Rick=Mercer}

8. Write a function reverse that accepts a dictionary from integers to strings as a parameter and returns a new dictionary of strings to integers that is the original's "reverse". The reverse of a dictionary is defined here to be a new dictionary that uses the values from the original as its keys and the keys from the original as its values. Since a dictionary’s values need not be unique but its keys must be, it is acceptable to have any of the original keys as the value in the result. In other words, if the original dictionary has pairs (*k1*, *v*) and (*k2*, *v*), the new dictionary must contain either the pair (*v*, *k1*) or (*v*, *k2*).

 For example, for the following dictionary:

{42=Marty, 81=Sue, 17=Ed, 31=Dave, 56=Ed, 3=Marty, 29=Ed}

 Your function could return the following new map (the order of the key/value pairs does not matter):

{Marty=3, Sue=81, Ed=29, Dave=31}

9. Write a function intersect that accepts two dictionaries from strings to integers as parameters and returns a new dictionary containing only the key/value pairs that exist in both of the parameter dictionaries. For a key/value pair to be included in your result, not only do both parameter maps need to contain a mapping for that key, but they need to map that key to the same value. For example, consider the following two dictionaries:

{Janet=87, Logan=62, Whitaker=46, Alyssa=100, Stefanie=80, Jeff=88, Kim=52, Sylvia=95}

{Logan=62, Kim=52, Whitaker=52, Jeff=88, Stefanie=80, Brian=60, Lisa=83, Sylvia=87}

 Calling your function on the preceding dictionaries would return the following new dictionary (the order of the key/value pairs does not matter):

{Logan=62, Stefanie=80, Jeff=88, Kim=52}

10. Write a function max\_occurrences that accepts a list of integers as a parameter and returns the number of times the most frequently occurring integer (the "mode") occurs in the list. Solve this problem using a dictionary as auxiliary storage. If the list is empty, return 0.

# Solutions

1.

def num\_unique(list):

 set = set()

 for i in range(0, len(list)):

 set.add(list[i])

 return len(set)

def num\_unique(list):

 set = set()

 for value in list:

 set.add(value)

 return len(set)

2.

def num\_in\_common(list1, list2)

 set1 = set()

 set2 = set()

 for value in list1):

 set1.add(value)

 for value in list2):

 set2.add(value)

 common = set1 & set2

 return common

3.

def max\_length(set):

 maximum = 0

 for s in set:

 max = max(maximum, len(s))

 return maximum

4.

def has\_odd(set):

 for value in set):

 if (value % 2 != 0):

 return True

 return False

5.

def contains3(list):

 counts = {}

 for value in list:

 if (value in counts):

 count = counts[value]

 count += 1

 counts[value] = count

 if (count >= 3):

 return True

 else:

 counts[value] = 1

 return False

# Solutions (continued)

6.

def counts(list, set):

 counts\_dict = {}

 for value in list:

 if (value in set):

 if (value in counts\_dict):

 counts\_dict[value] = counts\_dict[value] + 1

 else:

 counts\_dict[value] = 1

 return counts\_dict

7.

def is\_unique(dict):

 values = set()

 for value in dict.values():

 if (value in values):

 return False # duplicate

 else:

 values.add(value)

 return True

8.

def reverse(dict):

 result = {}

 for key in dict:

 value = dict[key]

 result[value] = key

 return result

9.

def intersect(dict1, dict2):

 result = {}

 for key in dict1:

 value = dict1[key]

 if (key in dict2 and value == dict2[key]):

 result[key] = value

 return result

10.

def max\_occurrences(list):

 counts = {}

 for value in list:

 if (value in counts):

 counts[value] = counts[value] + 1

 else:

 counts[value] = 1

 maximum = 0

 for count in counts.values():

 maximum = maximum(max, count)

 return maximum