Sets

A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference. Sets can also have important efficiency benefits.

One Motivation -- Lists can be slooooooow....

One motivation for using sets is that several important operations (adding an element, determining whether an element is in the set) take *constant time* regardless of the size of the set, rather than linear time in the size of the list.

In [ ]:
```
big_num = 10000000 # ten million
big_num_list = list(range(big_num))
big_num_set = set(big_num_list)
```

In [ ]:
```
small_num = 100
small_num_list = list(range(big_num - small_num, big_num))

# how many of small_num_list elements are in big_num_list?
import time
start = time.time()
count = 0
print("counting...")
for i in small_num_list:
    count = count + 1 if i in big_num_list else 0
end = time.time()
print("count using list:", count, "time:", end-start, "sec")
```

In [ ]:
```
# how many of small_num_list elements are in big_num_set?
start = time.time()
count = 0
## small_num_list = big_num_list
print("counting...")
for i in small_num_list:
    count = count + 1 if i in big_num_set else 0
end = time.time()
print("count using set:", count, "time:", end-start, "sec")

count_intersection = len(big_num_set.intersection(set(small_num_list)))
end2 = time.time()
print("count using set intersection:", count_intersection, "time:", end2-end, "sec")
```

Another Motivation -- Conceptual clarity with set operations

In [ ]:
```
# Lists can have duplicate elements, and lists are ordered
basket = ['apple', 'orange', 'apple', 'pear', 'orange']
print(basket)

# Creating a set from a list results in a set without duplicate elements
fruit1 = set(basket)
print(fruit1)
```

In [ ]:
```
# Adding the same element again to a set doesn't change the set
fruit1.add('apple')
print(fruit1)
```

In [ ]:
```
# But adding a different element does change (mutate) the set...
fruit1.add('banana')
print(fruit1)
```

In [ ]:
```
# Can discard/remove elements
fruit1.discard('grape')  #no exception if element not in set
fruit1.remove('apple') #exception if element not in set
print(fruit1)
```

In [ ]:
```
# Sets are unordered: cannot index or slice into a set
fruit1[0:]
```

In [ ]:
```
# Can iterate over the elements in a set, in loops or comprehensions
for elt in fruit1:
    if 'n' in elt:
        print(elt)

print([elt for elt in fruit1 if 'n' in elt])
print({elt for elt in fruit1 if 'n' in elt})
```

Basic set operations

In [ ]:
```
fruit2 = {'orange', 'apple', 'berry', 'grape', 'orange'}
print("fruit1 =", fruit1)
print("fruit2 =", fruit2)
```

In [ ]:
```
#Intersection
print("Intersection:", fruit1 & fruit2)

#Union
print("Union:", fruit1 | fruit2)

#Difference
print("Difference, fruit1 - fruit2:", fruit1 - fruit2)
print("Difference, fruit2 - fruit1:", fruit2 - fruit1)

#Symmetric Difference (all elts not in both)
print("Symmetric Difference:", fruit1 ^ fruit2)
```

Some set relations

In [ ]:
```
fruit3 = set()  #Create an empty set with 'set()' NOT with '{}'
fruit3.add('banana')
fruit3.add('pear')

print("fruit1 =", fruit1)
print("fruit2 =", fruit2)
print("fruit3 =", fruit3)
```

In [ ]:
```
#Subset
fruit3.issubset(fruit1)
```

In [ ]:
```
#Disjoint
fruit3.isdisjoint(fruit2)
```

In [ ]:
```
#Superset
fruit1.issuperset(fruit3)
```

What kind of objects can be in a set?

The elements of sets must be immutable hashable objects. Thus numbers, strings, tuples (as long as all elements of the tuple are also immutable/hashable objects) can be members of sets, but lists cannot be members of sets. And sets cannot be members of sets! (See frozensets if you're interested in an immutable/hashable variant of sets, that *can* be elements of a set.) The hashable restriction is what makes it possible to determine whether an element is in a set using constant time with respect to the size of the set; i.e., one does not need to iterate over all elements of a set to determine whether that element is in the set. (See 6.006 for more details on how this hashing works.)

In [ ]:
```
set(['a', 1])
```

In [ ]:
```
set(['a', [1]])
```

In [ ]:
```
set(['a', (1,)])
```

Practice

Given a list of integers `data`, return a new list `out` of boolean values such that `out[i] = True` if `data[i]` appears at least one more time in data, before index i.

In [ ]:
```
data = [7, 4, 7, 3]
# expected output: result = [False, False, True, False]
```

In [ ]:
```
def duplicate_view(data):
    pass

duplicate_view(data)
```

In [ ]:
```

```