Class 14

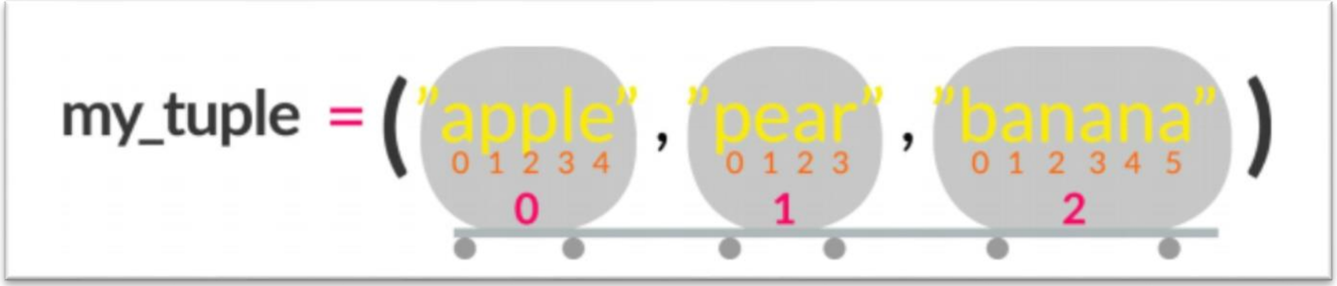
Week 7

Tuples

Tuple is actually a collection of things.

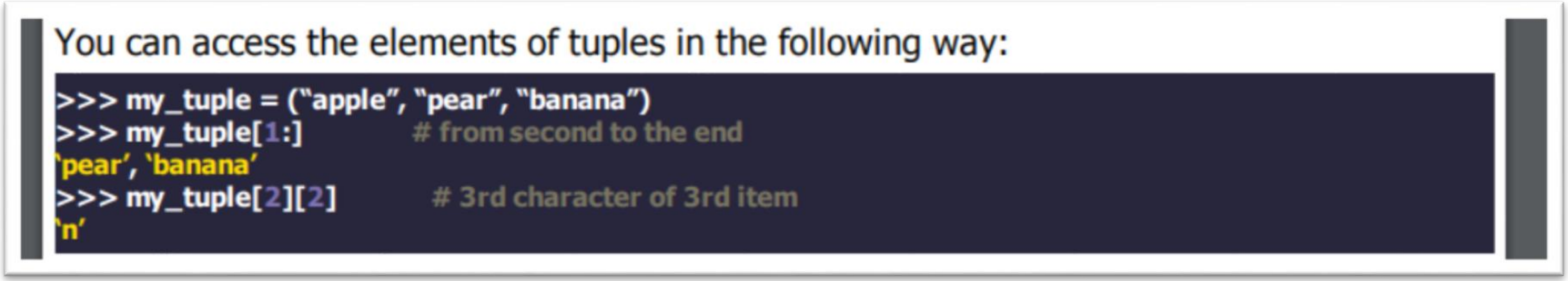
- It is unchangeable.
- It is presented by round brackets.

Tuples



```
my_tuple = ("apple", "pear", "banana")
```

Accessing Tuples

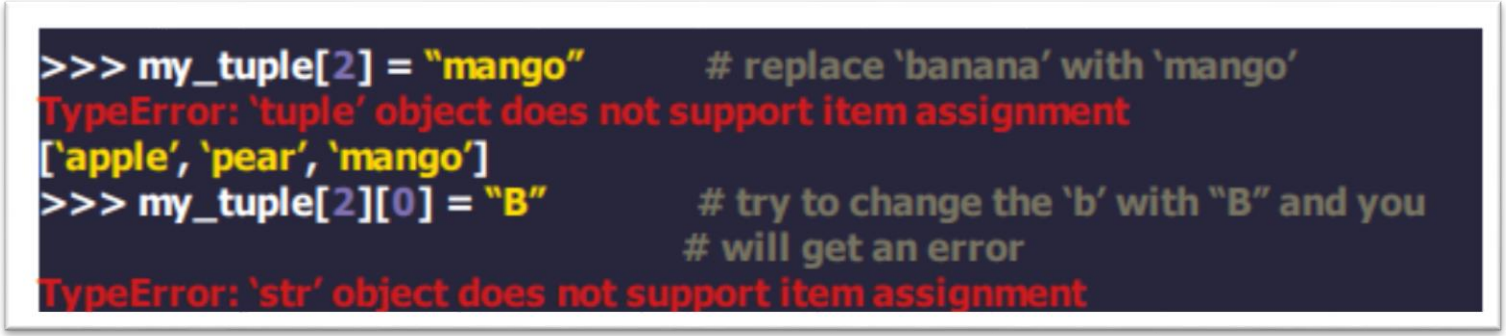


You can access the elements of tuples in the following way:

```
>>> my_tuple = (“apple”, “pear”, “banana”)
>>> my_tuple[1:]           # from second to the end
‘pear’, ‘banana’
>>> my_tuple[2][2]          # 3rd character of 3rd item
‘n’
```

Error While Changing

As Tuples can’t be changed.



```
>>> my_tuple[2] = “mango”          # replace ‘banana’ with ‘mango’
TypeError: ‘tuple’ object does not support item assignment
[‘apple’, ‘pear’, ‘mango’]
>>> my_tuple[2][0] = “B”           # try to change the ‘b’ with “B” and you
                                  # will get an error
TypeError: ‘str’ object does not support item assignment
```

BUT... WAIT

We can iterate through tuple, search in a tuple, del tuple, use len functions.

CODE → TUPLES

```
days = ("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday")


print(days)
print(days[1]) #access
days.count("Sunday") #counting
print(days.index("Sunday")) #getting index
print(len(days))
```

CODE→ TUPLES

```
for x in days: #applying for loop on tuples
    print(x)

if "Sunday" in days:
    print("Yes! Sunday is a Holiday“)
```

Convert List to Tuple

```
tuple(    )
vice versa → list()


fruits = tuple(["mango","banana","apple"])
print(fruits)
print(type(fruits))
```

Change Tuple to List --ADVANTAGE

Convert into list constructor and then we can apply all list operations

CODE → After converting to list

```
days = list(days)
days[0] = "Sunday"

days.pop(0)
days.insert(0,"Monday")

days = tuple(days)
print(days)

for day in range(0,len(days)):
    if days[day] == "Saturday":
        print("Saturday is holiay")
    else:
        print("not holiday")
```

Packing

```
a = 1, 2, 3 # a is the tuple (1, 2, 3)
       or
a = (1, 2, 3) # a is the tuple (1, 2, 3)
```

The assignment a = 1, 2, 3 is also called packing because it packs values together in a tuple.

Unpacking

```
# unpacking AKA multiple assignment
x, y, z = (1, 2, 3)
# x == 1
# y == 2
# z == 3
```

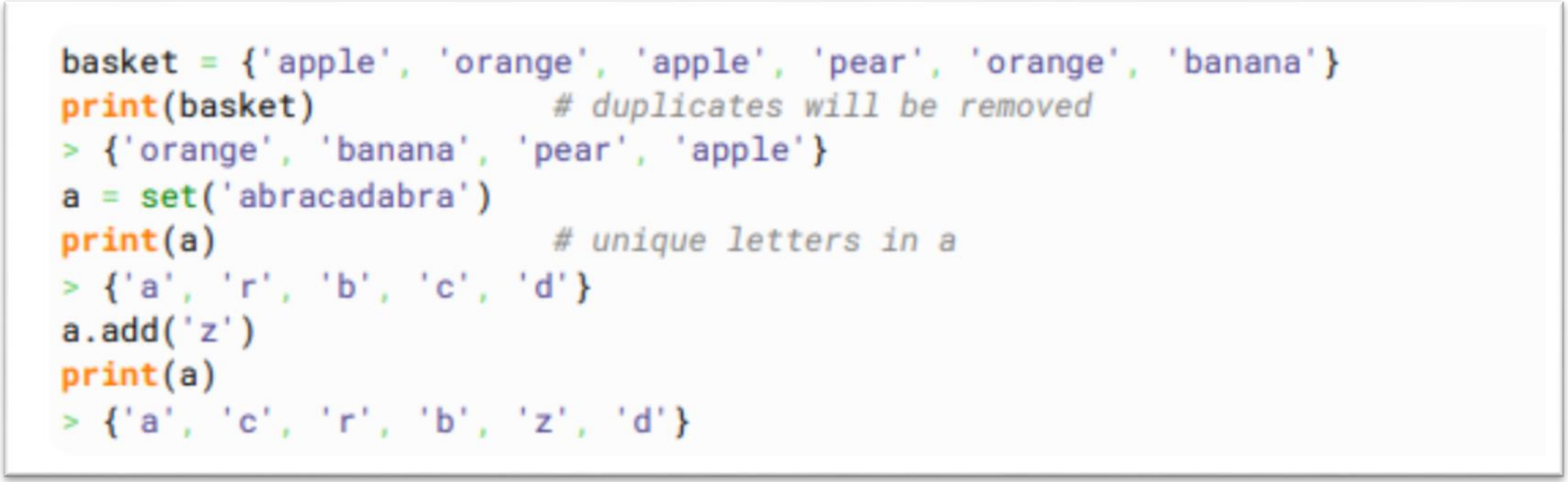
To unpack values from a tuple

SETS

Sets are unordered collections of unique objects, there are two types of sets.

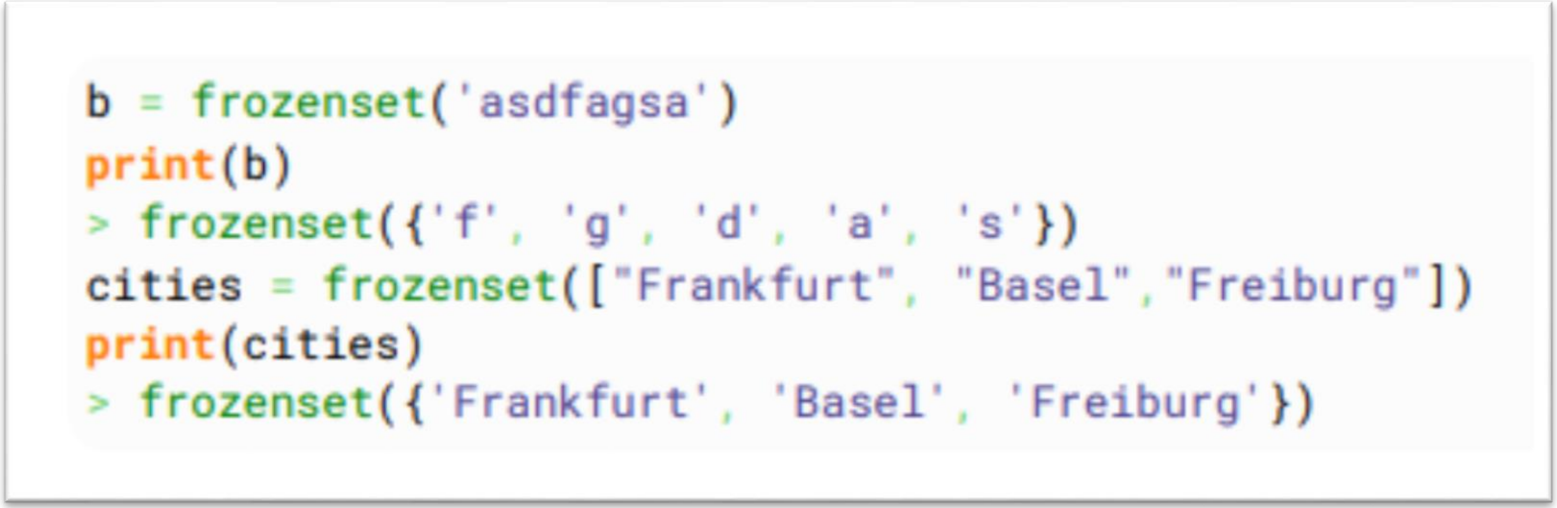
1. **Sets - They are mutable and new elements can be added once sets are defined.**
2. **Frozen Sets - They are immutable and new elements cannot added after its defined.**

SETS



```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)            # duplicates will be removed
> {'orange', 'banana', 'pear', 'apple'}
a = set('abracadabra')
print(a)                 # unique letters in a
> {'a', 'r', 'b', 'c', 'd'}
a.add('z')
print(a)
> {'a', 'c', 'r', 'b', 'z', 'd'}
```

Frozen Sets

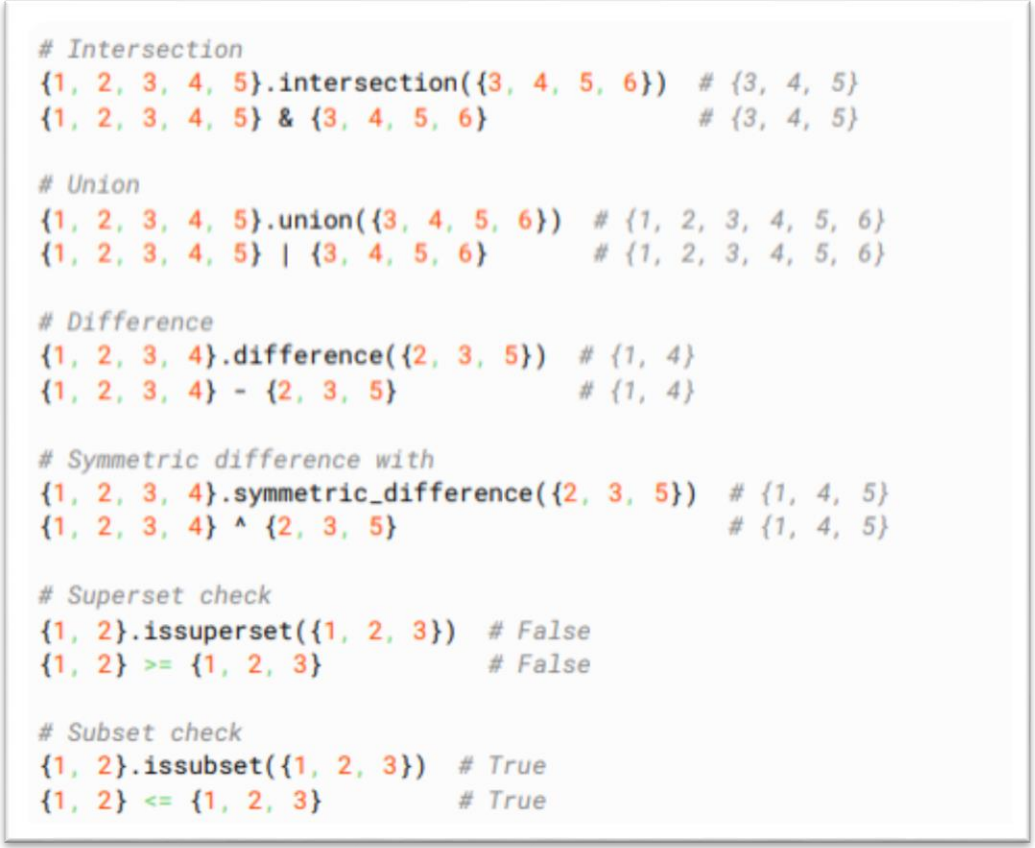


```
b = frozenset('asdfagsa')
print(b)
> frozenset({'f', 'g', 'd', 'a', 's'})
cities = frozenset(["Frankfurt", "Basel","Freiburg"])
print(cities)
> frozenset({'Frankfurt', 'Basel', 'Freiburg'})
```

Sets

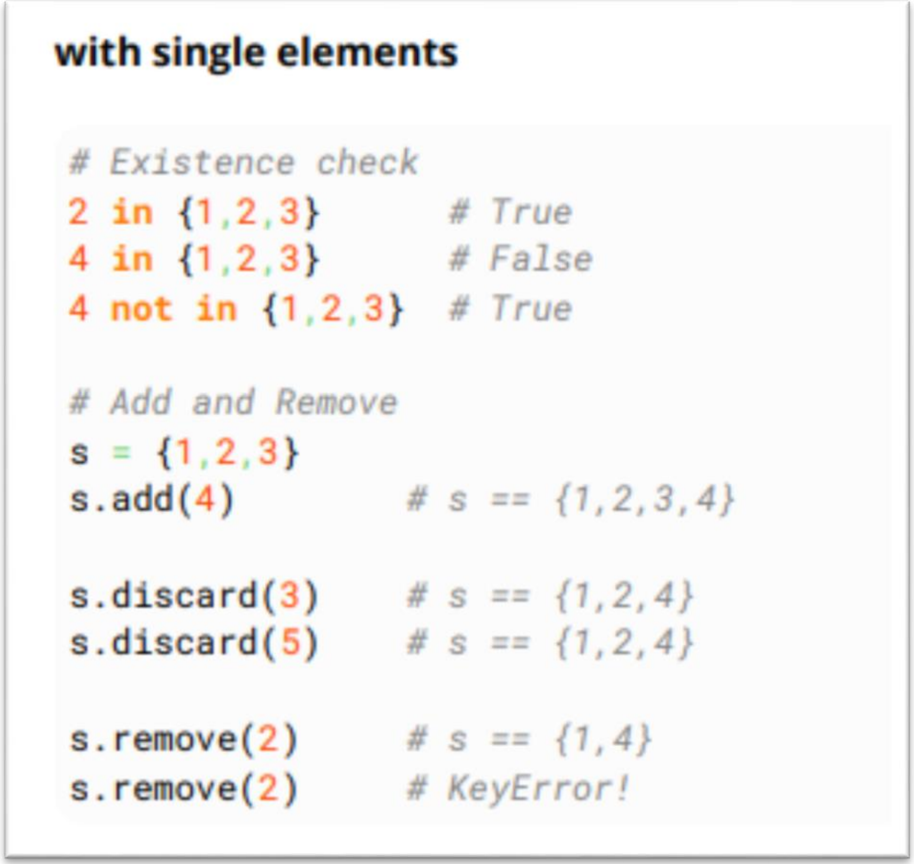
```
-  Distinct Values

y = set([2,4,5,6])
z = {2,6,7,8,9,10,11,12}
x = {2,4,5}
print(y | z)
print(y & z)
print(y - z)
print(y ^ z) #symmetric difference
print(y > z) # a superset
print(y < z) #a subset
print(x < y) # x is a subset of y
print(x > y) # x is not a superset
print(y < x)
print(y > x)
print(y.intersection(z))
print(y.union(z))
print(y.difference(z))
print(y.issubset(x))
print(y.issuperset(x))
```

Sets



```
# Intersection
{1, 2, 3, 4, 5}.intersection({3, 4, 5, 6})  # {3, 4, 5}
{1, 2, 3, 4, 5} & {3, 4, 5, 6}              # {3, 4, 5}

# Union
{1, 2, 3, 4, 5}.union({3, 4, 5, 6})  # {1, 2, 3, 4, 5, 6}
{1, 2, 3, 4, 5} | {3, 4, 5, 6}       # {1, 2, 3, 4, 5, 6}

# Difference
{1, 2, 3, 4}.difference({2, 3, 5})  # {1, 4}
{1, 2, 3, 4} - {2, 3, 5}            # {1, 4}

# Symmetric difference with
{1, 2, 3, 4}.symmetric_difference({2, 3, 5})  # {1, 4, 5}
{1, 2, 3, 4} ^ {2, 3, 5}                      # {1, 4, 5}

# Superset check
{1, 2}.issuperset({1, 2, 3})  # False
{1, 2} >= {1, 2, 3}           # False

# Subset check
{1, 2}.issubset({1, 2, 3})  # True
{1, 2} <= {1, 2, 3}         # True
```

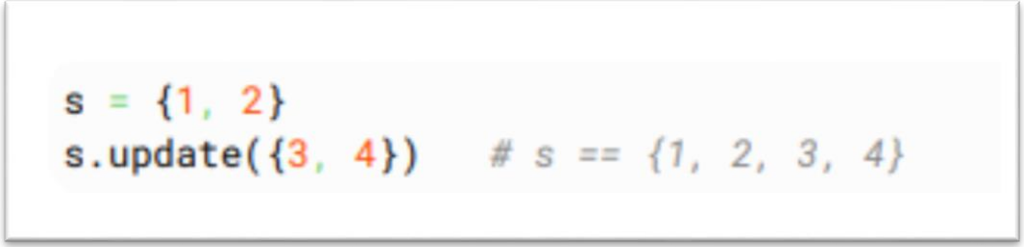
Sets



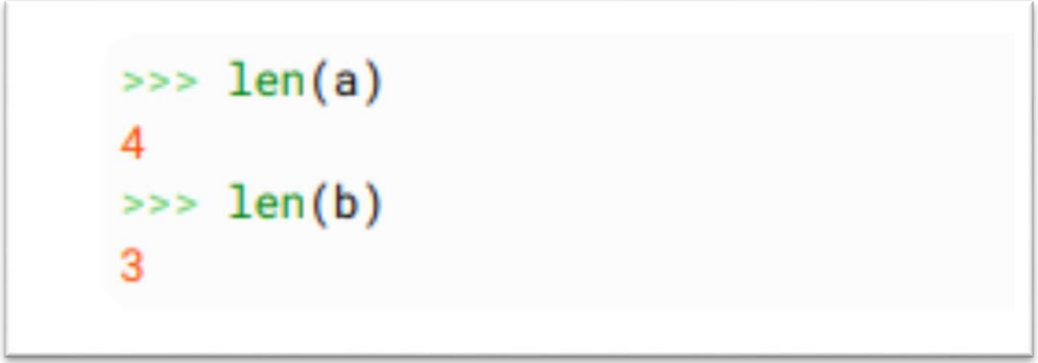
with single elements

```
# Existence check
2 in {1,2,3}        # True
4 in {1,2,3}        # False
4 not in {1,2,3}    # True

# Add and Remove
s = {1,2,3}
s.add(4)           # s == {1,2,3,4}

s.discard(3)       # s == {1,2,4}
s.discard(5)       # s == {1,2,4}

s.remove(2)        # s == {1,4}
s.remove(2)        # KeyError!
```

Sets

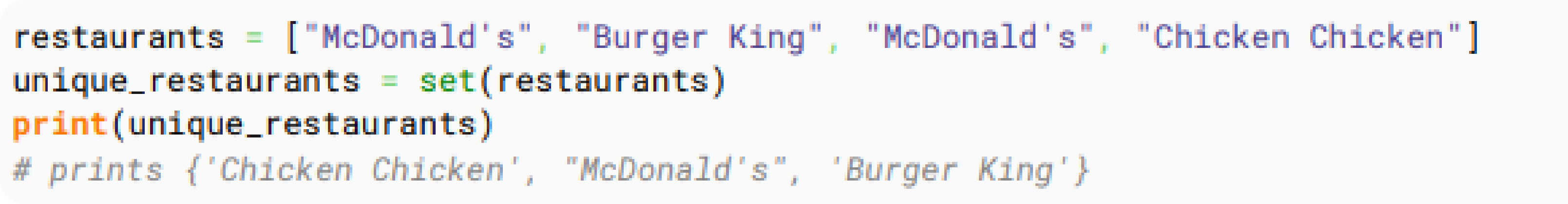


```
s = {1, 2}
s.update({3, 4})    # s == {1, 2, 3, 4}
```



```
>>> len(a)
4
>>> len(b)
3
```

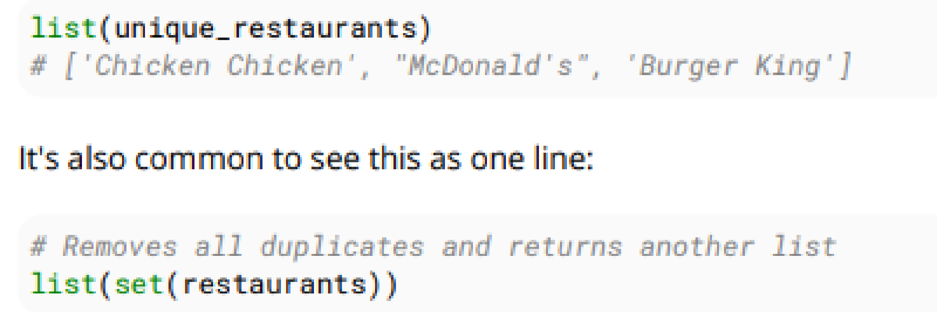
Let's say you've got a list of restaurants. You care about the unique restaurants in the list. The best way to get the unique elements from a list is to turn it into a set.



```
restaurants = ["McDonald's", "Burger King", "McDonald's", "Chicken Chicken"]
unique_restaurants = set(restaurants)
print(unique_restaurants)
# prints {'Chicken Chicken', "McDonald's", 'Burger King'}
```

Note that the set is not in the same order as the original list; that is because sets are unordered

This can easily be transformed back into a List with Python's built in list function, giving another list that is the same list as the original but without duplicates



```
list(unique_restaurants)
# ['Chicken Chicken', "McDonald's", 'Burger King']
```

It's also common to see this as one line:

```
# Removes all duplicates and returns another list
list(set(restaurants))
```