M110: Python Programming

Self-Study#3

Collection Data Types Dictionaries & Sets

AOU
الجامعة العربية المفتوحة
Arab Open University
Lebanon

Prepared by Dr. Ahmad Mikati

Contents

Introduction

- Python programming language has four collection data types- list, tuple, sets and dictionary. They have different characteristics based on the declaration and the usage.

- In this lecture, we will look at two of the fundamental additional types: Dictionaries and Sets.
- We will explore some of the operations that you may perform on these types, including ways to access and manipulate their contents.

Dictionaries

When you hear the word “dictionary,” you probably think about a large book containing words and their definitions. If you want to know the meaning of a particular word, you locate it in the dictionary to find its definition.

In Python, a ***dictionary*** is an object that stores a collection of data. Each element that is stored in a dictionary has two parts: a ***key*** and a ***value***. Key-value pairs are often referred to as mappings because each key is mapped to a value.

In fact, dictionary elements are commonly referred to as *key-value pairs*. When you want to retrieve a specific value from a dictionary, you use the key that is associated with that value.

An example would be a program that lets us enter a person’s name and gives us that person’s phone number. The program could use a dictionary in which each element contains a person’s name as the key, and that person’s phone number as the value. If we know a person’s name, then we can retrieve that person’s phone number.

Creating a Dictionary

You can create a dictionary by enclosing the elements inside a set of curly braces (**{}**).
An element consists of a key, followed by a colon (**:**), followed by a value. The elements are separated by commas.

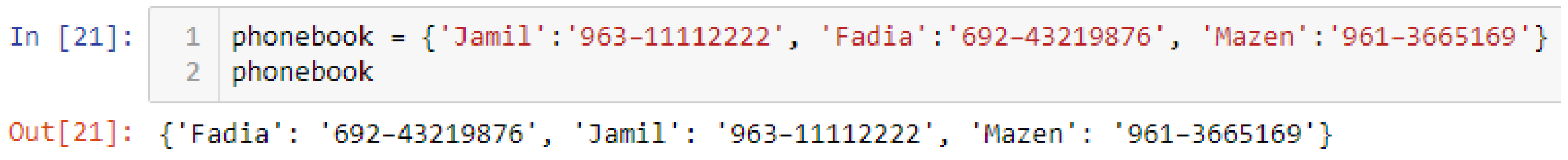
```
phonebook = {'Jamil':'963−11112222', 'Fadia':'692−43219876',
'Mazen':'961−3665169'}
```

In the above example, the keys and the values are strings. The values in a dictionary can be objects of any type, but the keys must be immutable objects.
For example, keys can be strings, integers, floating-point values, or tuples.
Keys cannot be lists or any other type of mutable objects.

Dictionaries themselves are mutable, so entries can be added, removed, and changed at any time.
Note, though, that because entries are accessed by their key, we can't have two entries with the same key.

Retrieving a Value from a Dictionary

The elements in a dictionary are not stored in any particular order. For example, look at the following interactive session in which a dictionary is created, and its elements are displayed:



```
In [21]:  phonebook = {'Jamil':'963-11112222', 'Fadia':'692-43219876', 'Mazen':'961-3665169'}
          phonebook

Out[21]: {'Fadia': '692-43219876', 'Jamil': '963-11112222', 'Mazen': '961-3665169'}
```

Notice the order in which the elements are displayed is different than the order in which they were created.

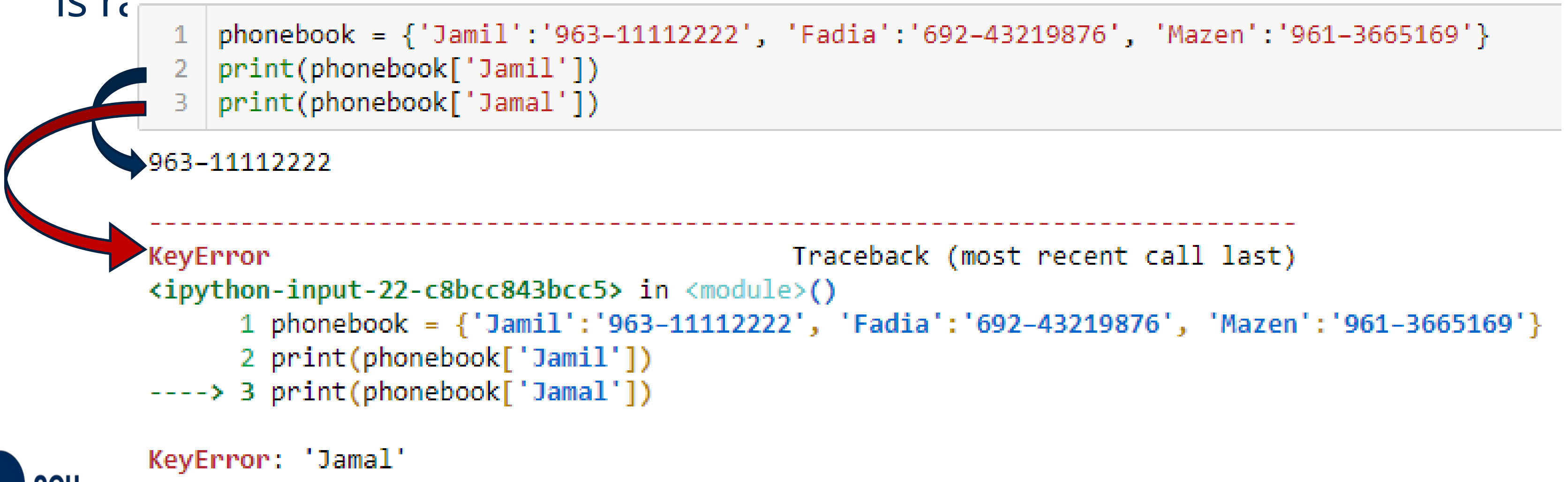
<u>This illustrates how dictionaries are not sequences</u>, like lists, tuples, and strings.
As a result, <u>you cannot use a numeric index to retrieve a value by its position from a dictionary</u>.
Instead, you use a key to retrieve a value.

Retrieving a Value from a Dictionary

To retrieve a value from a dictionary, you simply write an expression in the following general format:

dictionary_name[*key*]

In the general format, *dictionary_name* is the variable that references the dictionary, and *key* is a key.
If the key exists in the dictionary, the expression returns the value that is associated with the key. If the key does not exist, a *KeyError* exception is ra



```python
phonebook = {'Jamil':'963-11112222', 'Fadia':'692-43219876', 'Mazen':'961-3665169'}
print(phonebook['Jamil'])
print(phonebook['Jamal'])
```

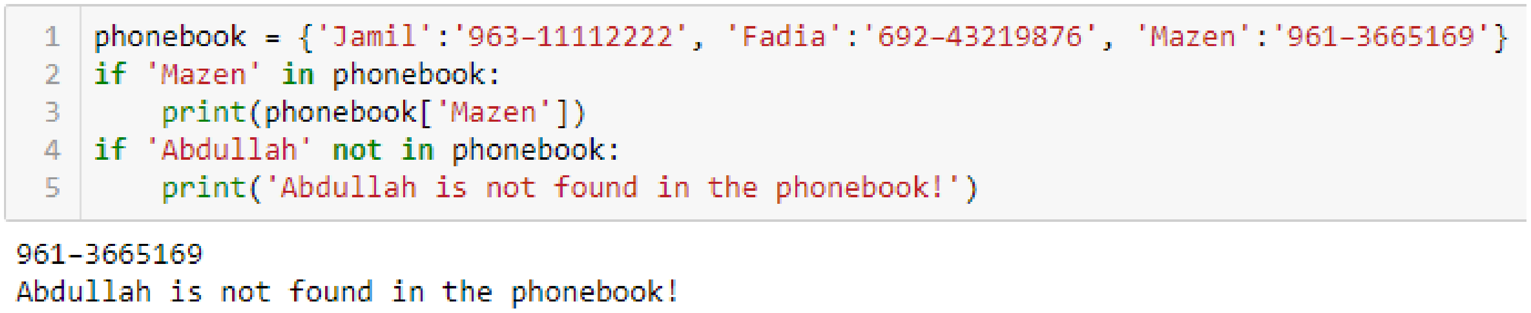
```
963-11112222

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-22-c8bcc843bcc5> in <module>()
      1 phonebook = {'Jamil':'963-11112222', 'Fadia':'692-43219876', 'Mazen':'961-3665169'}
      2 print(phonebook['Jamil'])
----> 3 print(phonebook['Jamal'])

KeyError: 'Jamal'
```

Using the in and not in Operators to Test for a Value in a Dictionary

As previously demonstrated, a ***KeyError*** exception is raised if you try to retrieve a value from a dictionary using a nonexistent key.

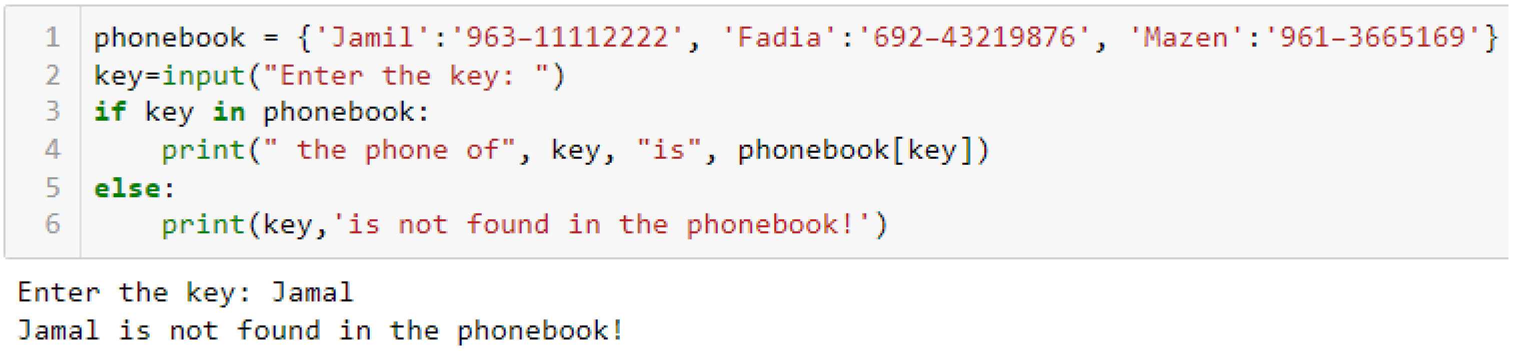
To prevent such an exception, you can use the **in** operator to determine whether a key exists before you try to use it to retrieve a value.

Yo
n



```python
phonebook = {'Jamil':'963-11112222', 'Fadia':'692-43219876', 'Mazen':'961-3665169'}
if 'Mazen' in phonebook:
    print(phonebook['Mazen'])
if 'Abdullah' not in phonebook:
    print('Abdullah is not found in the phonebook!')
```

```
961-3665169
Abdullah is not found in the phonebook!
```



```python
phonebook = {'Jamil':'963-11112222', 'Fadia':'692-43219876', 'Mazen':'961-3665169'}
key=input("Enter the key: ")
if key in phonebook:
    print(" the phone of", key, "is", phonebook[key])
else:
    print(key,'is not found in the phonebook!')
```

```
Enter the key: Jamal
Jamal is not found in the phonebook!
```

N.B: Keep in mind that string comparisons with the **in** and **not in** operators are case sensitive!

Adding Elements to a Dictionary

You can add a key-value pair to a dictionary using a new index key and assigning a value to it:

```
# adding to/updating a dictionary
phonebook = {'Jamil':'963−11112222', 'Fadia':'692−43219876',
'Mazen':'961−3665169'}
phonebook['Jamal']='961-76411262'
phonebook['Fadia']='962-43219875’
print(phonebook)
```

```
{'Jamil': '963−11112222', 'Fadia': '962-43219875', 'Mazen':
'961−3665169', 'Jamal': '961-76411262'}
```

Deleting Elements from a Dictionary

You can delete an existing key-value pair from a dictionary with the **del** statement. Below is the general format:

del dictionary_name[key]

After the statement executes, the *key* and its associated value will be deleted
from the dictionary. If the *key* does not exist, a KeyError exception is raised.

```
phonebook = {'Jamil':'963−11112222', 'Fadia':'692−43219876',
'Mazen':'961−3665169'}
del phonebook['Fadia’]
print(phonebook)
```

```
{'Jamil': '963−11112222', 'Mazen':
'961−3665169'}
```

More about Dictionaries

Finding the Number of Elements in a Dictionary

You can use the built-in len function to get the number of elements in a dictionary.

phonebook = {'Jamil':'963−11112222', 'Fadia':'692−43219876', 'Mazen':'961−3665169'}
print(len(phonebook)) ⟶ **3**

Creating an Empty Dictionary

You can use an empty set of curly braces to create an empty dictionary **{}**.

Mixing Data Types in a Dictionary

The *keys* in a dictionary must be immutable objects, but their associated values can be any type of object. The keys can be of different types, too, as long as they are immutable.

Key types: string, integer, tuple

mixed_types = {'dani':1, 2022:"Elections", (1, 2, 3):[3, 6, 9]}

Value types: integer, string, list

Useful Dictionary Methods/Functions

Method	Description
clear	Clears the contents of a dictionary.
get	Gets the value associated with a specified key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.
items	Returns all the keys in a dictionary and their associated values as a sequence of tuples.
keys	Returns all the keys in a dictionary as a sequence of tuples.
pop	Returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.
popitem	Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.

Function	Description
in	returns True if the dictionary contains an entry with the tested key
len	returns the number of elements in a dictionary
zip	allows you to combine two or more iterables, such as lists, tuples, or dictionaries, into a single iterable.

Examples

```
# Create two lists
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 28]
# Create a dictionary using zip()
people = dict(zip(names, ages))
print("Original dictionary:", people)
```

```
Output: {'Alice': 25, 'Bob': 30, 'Charlie':
28}
```

```
# Clear the dictionary
people.clear()
print("Dictionary after clearing:", people)
```

```
Output: {}
```

Arab Open University Lebanon

Examples

```
people={'David': 35, 'Emily': 22}
# Get all values as a list
values_list = list(people.values())
print("List of values:", values_list)
```

Output: [35, 22]

```
# Get the value for a specific key
age_of_bob = people.get("Bob")
print("Age of Bob:", age_of_bob)
```

Output: None (since "Bob" is not in the dictionary)

```
# Get the value for a specific key, with a default value if not found
age_of_david = people.get("David", "Not found")
print("Age of David:", age_of_david)
```

Output: 35

Sets

A set is an object that stores a collection of data in the same way as mathematical sets.

There are some aspects that you should know about sets:

- All the elements in a set must be unique. No two elements can have the same value.
- Sets are unordered, which means that the elements in a set are not stored in any order.
- Stored elements in a set can be of different data types.

Sets

Creating a Set

To create a set, you need to call the built-in set function.
For example, my_set = set() to create an empty set;

The argument that you pass must be an object that contains iterable elements, such as a list, a tuple, or a string.
The individual elements of the object that you pass as an argument become elements of the set.

```
set1=set(['x', 'y', 'z'])
print(set1)
```

{'y', 'x', 'z'} a set containing the elements ‘x', ‘y', and ‘z'.

```
set2=set('ahmad')
print(set2)
```

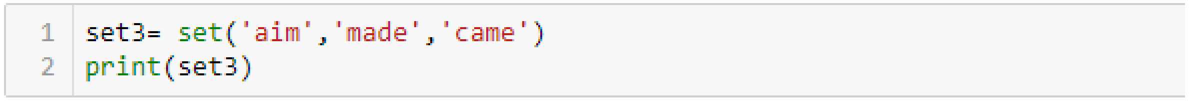
{'a', 'h', 'm', 'd'} Sets cannot contain duplicate elements.

What if you want to create a set in which each element is a string containing more than one character?

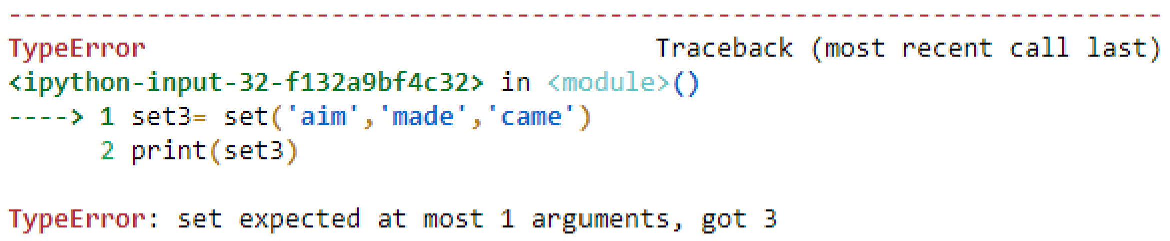
Sets

Creating a Set

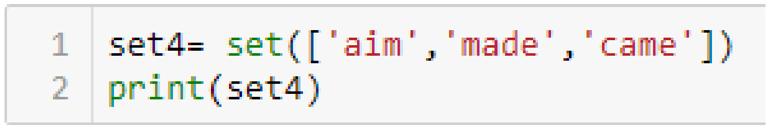
If you want to create a set in which each element is a string containing more than one character, you should pass a **list** containing the strings



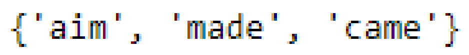
```python
set3= set('aim','made','came')
print(set3)
```



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-f132a9bf4c32> in <module>()
----> 1 set3= set('aim','made','came')
      2 print(set3)

TypeError: set expected at most 1 arguments, got 3
```



```python
set4= set(['aim','made','came'])
print(set4)
```

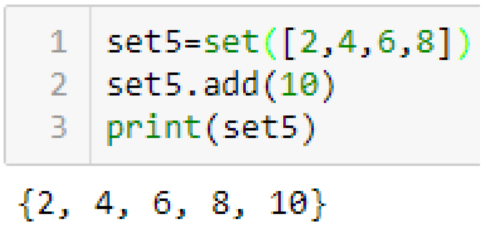


```
{'aim', 'made', 'came'}
```

Sets

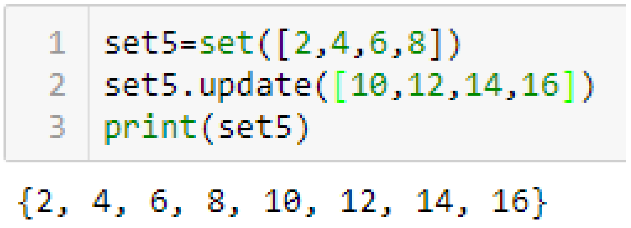
Adding, updating, and Removing Elements

Sets are mutable objects, so you can add items to them, update the sets, and remove items from them.
You use the ***add*** method to add an element to a set.
You can add a group of elements to a set all at one time with the ***update*** method.
You can remove an item from a set with either the ***remove*** method or the ***discard*** method.



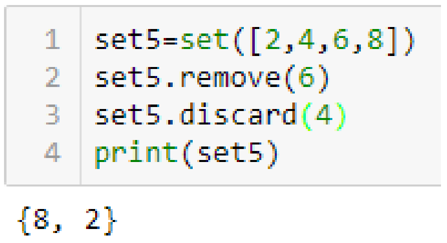
```
set5=set([2,4,6,8])
set5.add(10)
print(set5)
```

```
{2, 4, 6, 8, 10}
```



```
set5=set([2,4,6,8])
set5.update([10,12,14,16])
print(set5)
```

```
{2, 4, 6, 8, 10, 12, 14, 16}
```



```
set5=set([2,4,6,8])
set5.remove(6)
set5.discard(4)
print(set5)
```

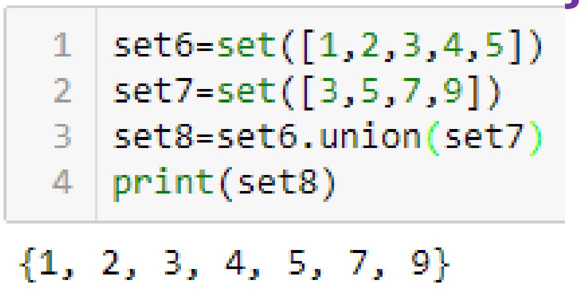
```
{8, 2}
```

Sets

Finding the Union and Intersection of Sets

The union of two sets is a set that contains all the elements of both sets, while the intersection of two sets is a set that contains only the elements that are found in both sets.

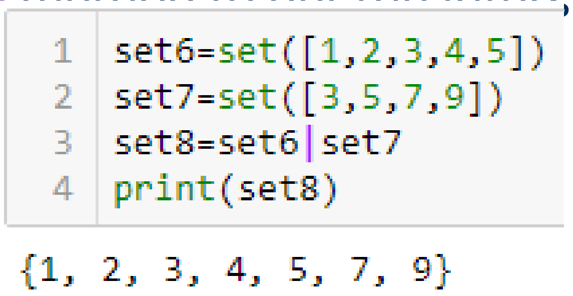
In Python, you can call the ***union*** method (or |) to get the union of two sets.
Moreover, you can call the ***intersection*** method to get the intersection of two se



```python
set6=set([1,2,3,4,5])
set7=set([3,5,7,9])
set8=set6.union(set7)
print(set8)
```

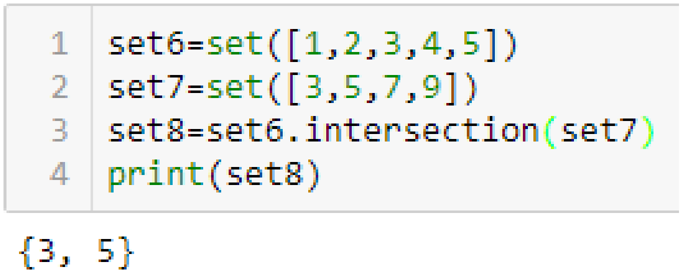
```
{1, 2, 3, 4, 5, 7, 9}
```

or



```python
set6=set([1,2,3,4,5])
set7=set([3,5,7,9])
set8=set6|set7
print(set8)
```

```
{1, 2, 3, 4, 5, 7, 9}
```



```python
set6=set([1,2,3,4,5])
set7=set([3,5,7,9])
set8=set6.intersection(set7)
print(set8)
```

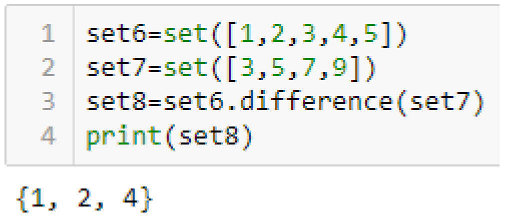
```
{3, 5}
```

Sets

Finding the Difference and Symmetric Difference of Sets

The **difference** of set1 and set2 is the elements that appear in set1 but do not appear in set2, while the **symmetric difference** of two sets is a set that contains the elements that are not shared by the sets.

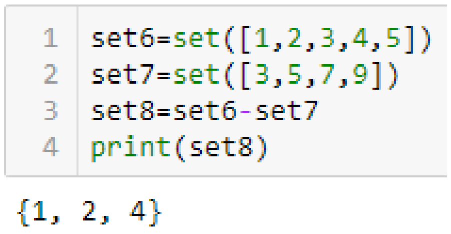
You can call the ***difference*** method (or -)to get the difference of two sets. You can call the ***symmetric_difference*** method (or ^)to get the symmetric difference of two sets.



```
set6=set([1,2,3,4,5])
set7=set([3,5,7,9])
set8=set6.difference(set7)
print(set8)
```

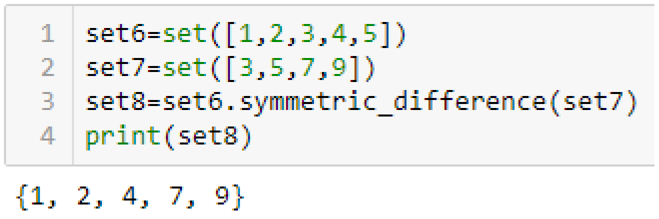
```
{1, 2, 4}
```

or



```
set6=set([1,2,3,4,5])
set7=set([3,5,7,9])
set8=set6-set7
print(set8)
```

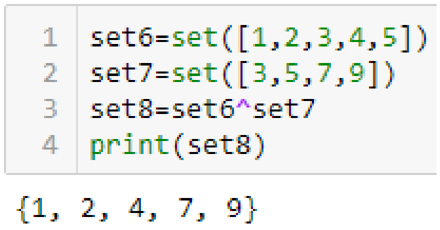
```
{1, 2, 4}
```



```
set6=set([1,2,3,4,5])
set7=set([3,5,7,9])
set8=set6.symmetric_difference(set7)
print(set8)
```

```
{1, 2, 4, 7, 9}
```

or



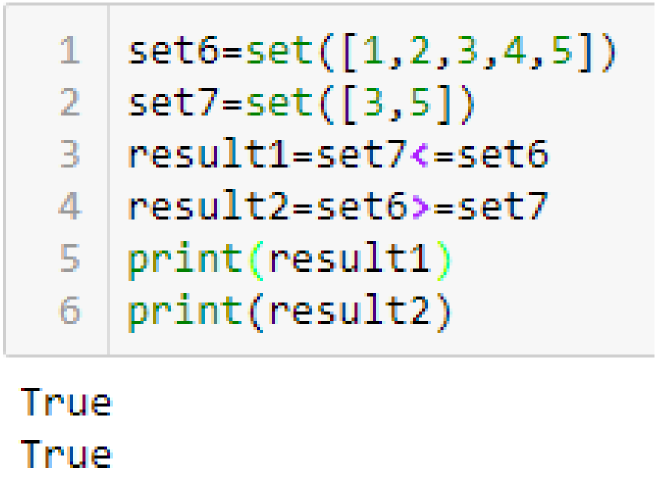
```
set6=set([1,2,3,4,5])
set7=set([3,5,7,9])
set8=set6^set7
print(set8)
```

```
{1, 2, 4, 7, 9}
```

Sets

Finding Subsets and Supersets

In Python, you can call the ***issubset*** method (or **<=**) to determine whether one set is a subset of another. The method returns True if set2 is a subset of set1. Otherwise, it returns False.
Also, you can call the ***issuperset*** method (or **>=**) to determine whether one set is a superset of another. The method returns True if set1 is a superset
of set2.



```python
set6=set([1,2,3,4,5])
set7=set([3,5])
result1=set7<=set6
result2=set6>=set7
print(result1)
print(result2)
```

```
True
True
```

Summary

This lecture covered:

Dictionaries, including:

- Creating dictionaries
- Inserting, retrieving, adding, and deleting key-value pairs
- Dictionary methods

Sets, including:

- Creating sets
- Adding elements to and removing elements from sets
- Finding set union, intersection, difference and symmetric difference
- Finding subsets and supersets