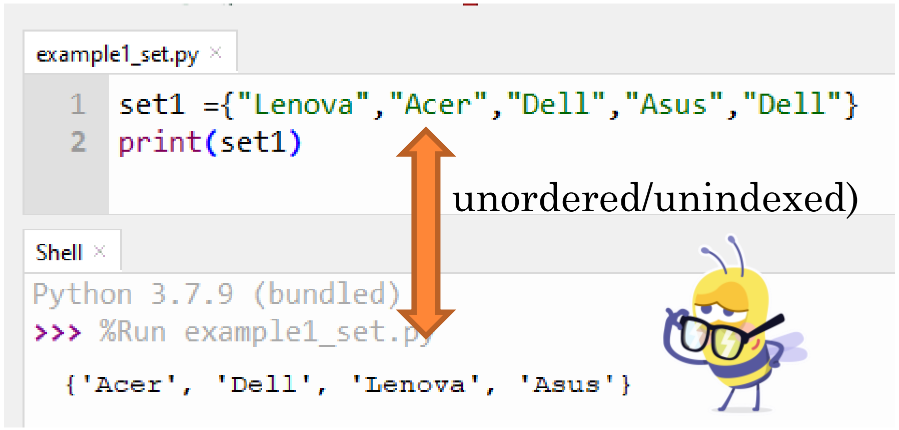
Data structures in Python

Week 8A: Sets

Sets

It is used to store multiple items in a single variable.
A set is a collection of *unordered*, *unchangeable*, and *unindexed* + *no duplicates values.*
The elements for set is defined within { }

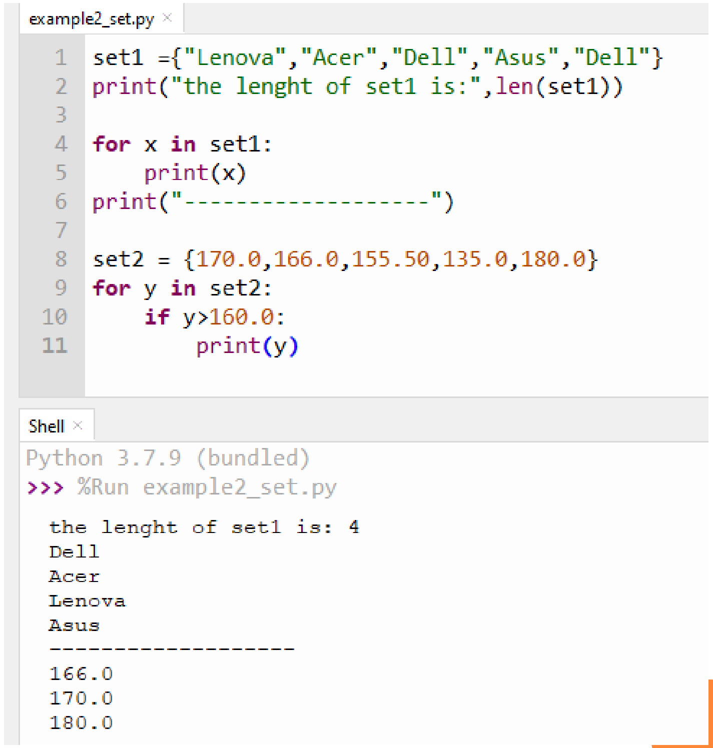


example1_set.py

```
set1 ={"Lenova","Acer","Dell","Asus","Dell"}
print(set1)
```

Shell

```
Python 3.7.9 (bundled)
>>> %Run example1_set.py

 {'Acer', 'Dell', 'Lenova', 'Asus'}
```

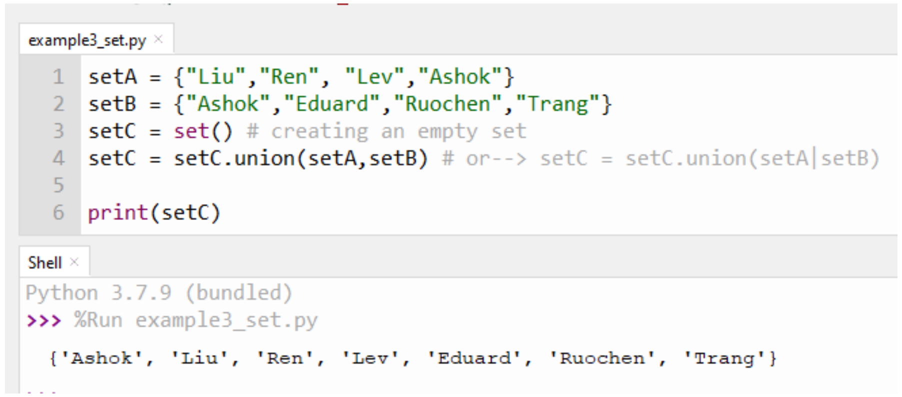


example2_set.py

```
set1 ={"Lenova","Acer","Dell","Asus","Dell"}
print("the lenght of set1 is:",len(set1))

for x in set1:
    print(x)
print("-------------------")

set2 = {170.0,166.0,155.50,135.0,180.0}
for y in set2:
    if y>160.0:
        print(y)
```

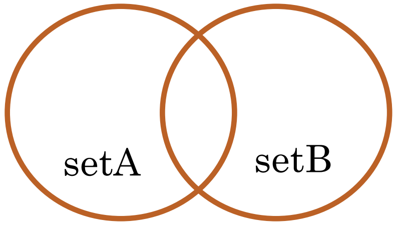
Shell

```
Python 3.7.9 (bundled)
>>> %Run example2_set.py

 the lenght of set1 is: 4
 Dell
 Acer
 Lenova
 Asus
 -------------------
 166.0
 170.0
 180.0
```

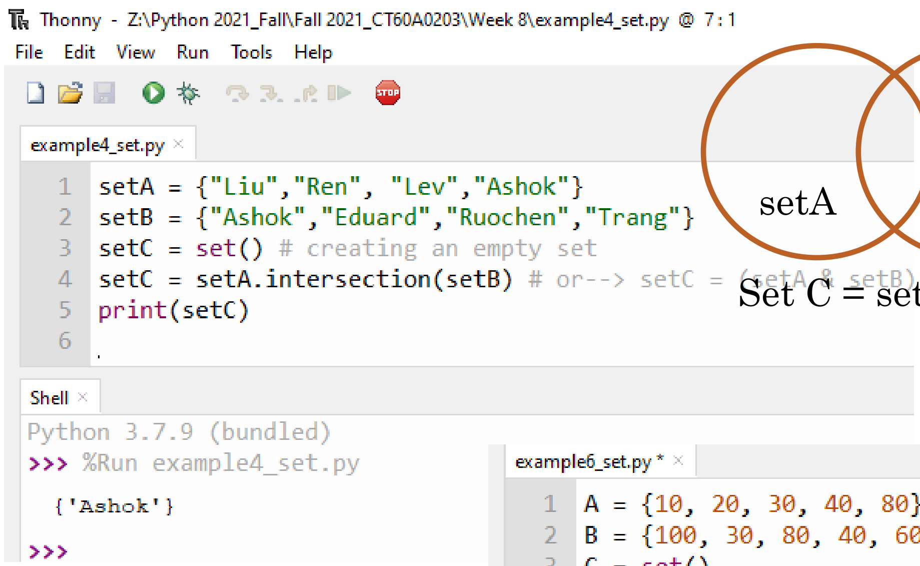


```python
setA = {"Liu","Ren", "Lev","Ashok"}
setB = {"Ashok","Eduard","Ruochen","Trang"}
setC = set() # creating an empty set
setC = setC.union(setA,setB) # or--> setC = setC.union(setA|setB)

print(setC)
```

```
Python 3.7.9 (bundled)
>>> %Run example3_set.py

 {'Ashok', 'Liu', 'Ren', 'Lev', 'Eduard', 'Ruochen', 'Trang'}
```

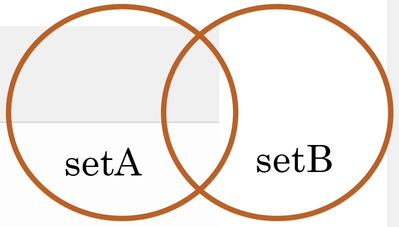


Set C = setA∪ setB

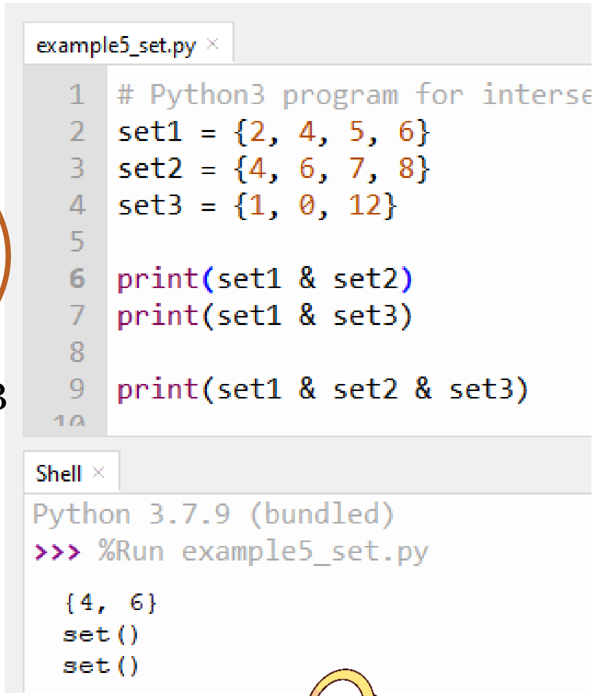

Thonny - Z:\Python 2021_Fall\Fall 2021_CT60A0203\Week 8\example4_set.py @ 7 : 1
File Edit View Run Tools Help
example4_set.py
Shell

```python
setA = {"Liu","Ren", "Lev","Ashok"}
setB = {"Ashok","Eduard","Ruochen","Trang"}
setC = set() # creating an empty set
setC = setA.intersection(setB) # or--> setC = (setA & setB)
print(setC)
```

```
Python 3.7.9 (bundled)
>>> %Run example4_set.py

 {'Ashok'}

>>>
```

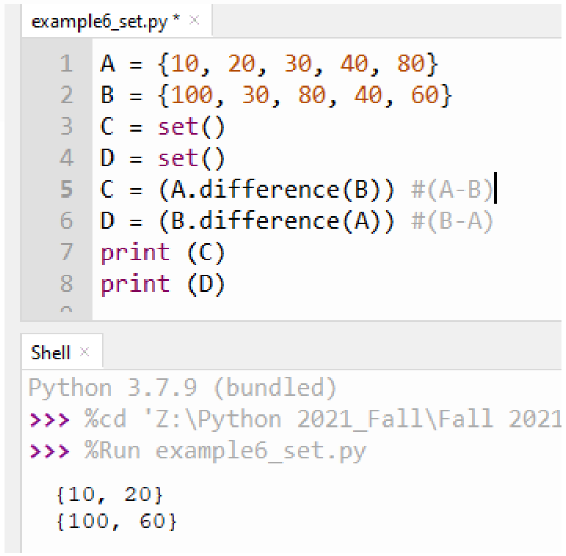


Set C = setA ∩ setB



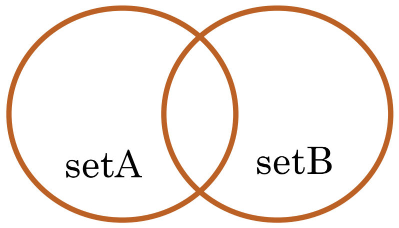
```python
# Python3 program for interse
set1 = {2, 4, 5, 6}
set2 = {4, 6, 7, 8}
set3 = {1, 0, 12}

print(set1 & set2)
print(set1 & set3)

print(set1 & set2 & set3)
```

```
Python 3.7.9 (bundled)
>>> %Run example5_set.py

 {4, 6}
 set()
 set()
```

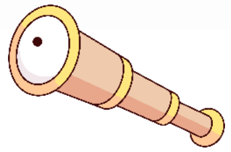


```python
A = {10, 20, 30, 40, 80}
B = {100, 30, 80, 40, 60}
C = set()
D = set()
C = (A.difference(B)) #(A-B)
D = (B.difference(A)) #(B-A)
print (C)
print (D)
```

```
Python 3.7.9 (bundled)
>>> %cd 'Z:\Python 2021_Fall\Fall 2021
>>> %Run example6_set.py

 {10, 20}
 {100, 60}
```



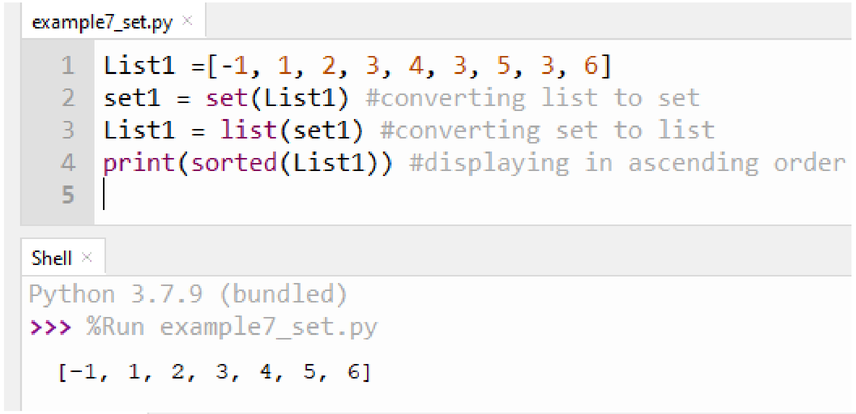
Set C = setA - setB



Set theory

- Union - ∪
- Intersection, and
- Difference -

Let’s try some tasks where the use of sets will be handy



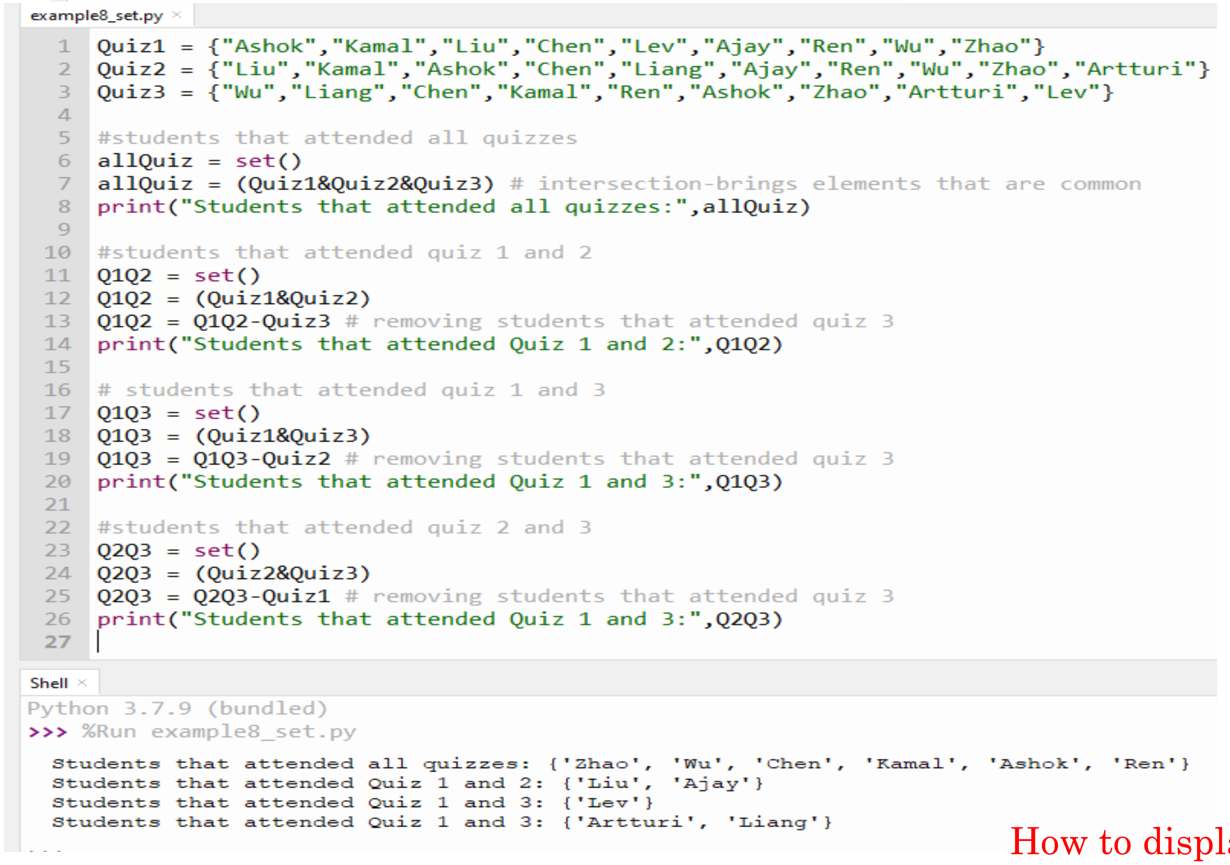
example7_set.py

```python
List1 =[-1, 1, 2, 3, 4, 3, 5, 3, 6]
set1 = set(List1) #converting list to set
List1 = list(set1) #converting set to list
print(sorted(List1)) #displaying in ascending order
```

Shell

```
Python 3.7.9 (bundled)
>>> %Run example7_set.py

 [-1, 1, 2, 3, 4, 5, 6]
```

Suppose List1 =[-1, 1, 2, 3, 4, 3, 5, 3, 6] and want to remove duplicates from it (exercise 6 of Week 7).



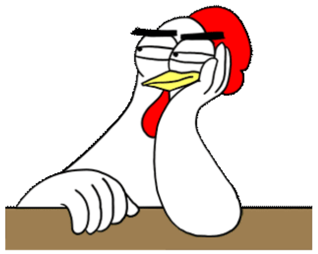
example8_set.py

```python
Quiz1 = {"Ashok","Kamal","Liu","Chen","Lev","Ajay","Ren","Wu","Zhao"}
Quiz2 = {"Liu","Kamal","Ashok","Chen","Liang","Ajay","Ren","Wu","Zhao","Artturi"}
Quiz3 = {"Wu","Liang","Chen","Kamal","Ren","Ashok","Zhao","Artturi","Lev"}

#students that attended all quizzes
allQuiz = set()
allQuiz = (Quiz1&Quiz2&Quiz3) # intersection-brings elements that are common
print("Students that attended all quizzes:",allQuiz)

#students that attended quiz 1 and 2
Q1Q2 = set()
Q1Q2 = (Quiz1&Quiz2)
Q1Q2 = Q1Q2-Quiz3 # removing students that attended quiz 3
print("Students that attended Quiz 1 and 2:",Q1Q2)

# students that attended quiz 1 and 3
Q1Q3 = set()
Q1Q3 = (Quiz1&Quiz3)
Q1Q3 = Q1Q3-Quiz2 # removing students that attended quiz 3
print("Students that attended Quiz 1 and 3:",Q1Q3)

#students that attended quiz 2 and 3
Q2Q3 = set()
Q2Q3 = (Quiz2&Quiz3)
Q2Q3 = Q2Q3-Quiz1 # removing students that attended quiz 3
print("Students that attended Quiz 1 and 3:",Q2Q3)
```

Shell

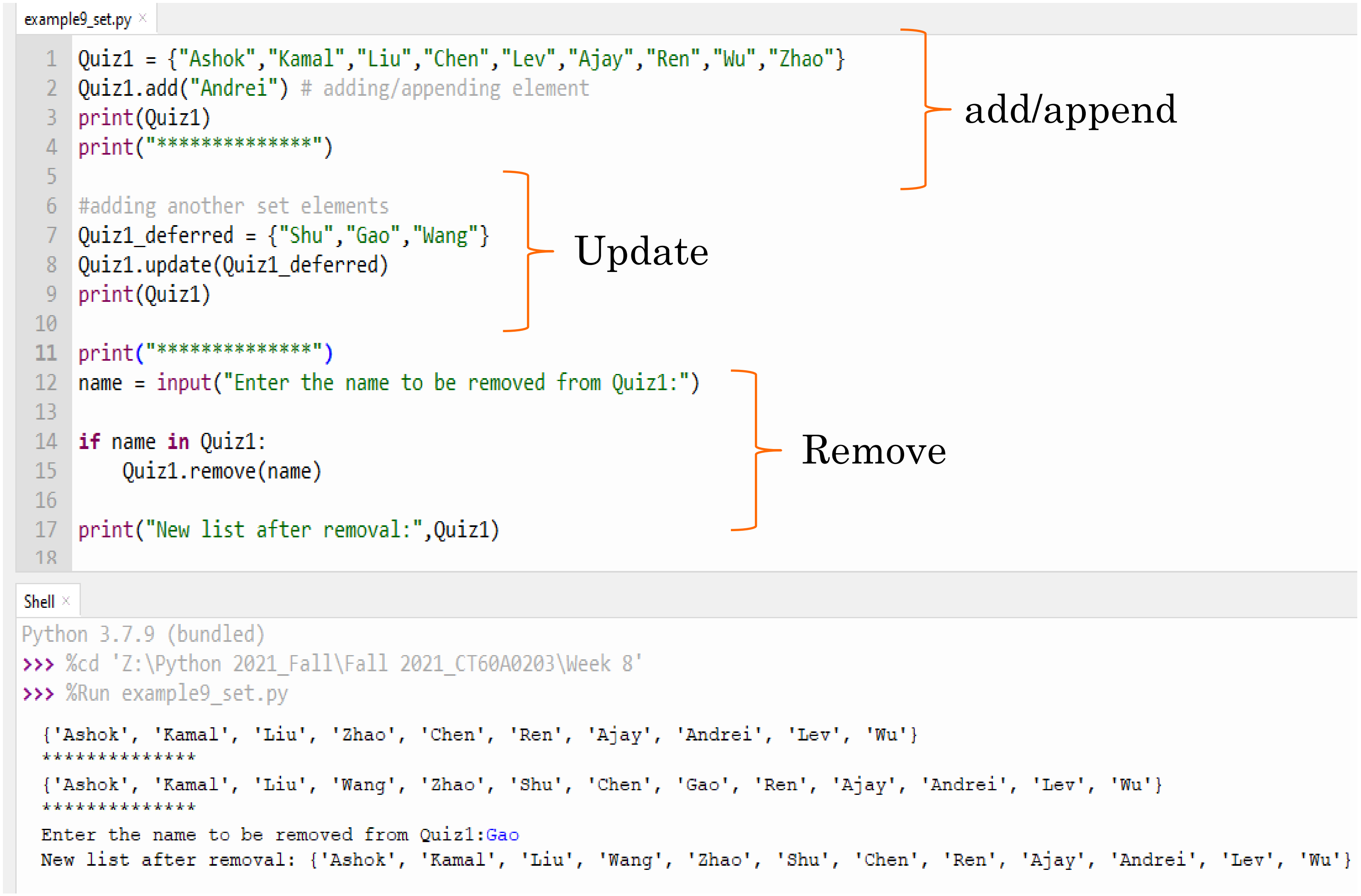
```
Python 3.7.9 (bundled)
>>> %Run example8_set.py

 Students that attended all quizzes: {'Zhao', 'Wu', 'Chen', 'Kamal', 'Ashok', 'Ren'}
 Students that attended Quiz 1 and 2: {'Liu', 'Ajay'}
 Students that attended Quiz 1 and 3: {'Lev'}
 Students that attended Quiz 1 and 3: {'Artturi', 'Liang'}
```

There are 3 sets that contain names of students who attended quiz1, quiz2, and quiz3. Teacher wants to get the list of students that attended all quizzes, attended only quiz1 and quiz2, quiz1 and quiz3, and quiz2 and quiz3.



How to display these in ascending order?

How to add, update, and delete elements of set?



example9_set.py

```
Quiz1 = {"Ashok","Kamal","Liu","Chen","Lev","Ajay","Ren","Wu","Zhao"}
Quiz1.add("Andrei") # adding/appending element
print(Quiz1)
print("**************")

#adding another set elements
Quiz1_deferred = {"Shu","Gao","Wang"}
Quiz1.update(Quiz1_deferred)
print(Quiz1)

print("**************")
name = input("Enter the name to be removed from Quiz1:")

if name in Quiz1:
    Quiz1.remove(name)

print("New list after removal:",Quiz1)
```

add/append

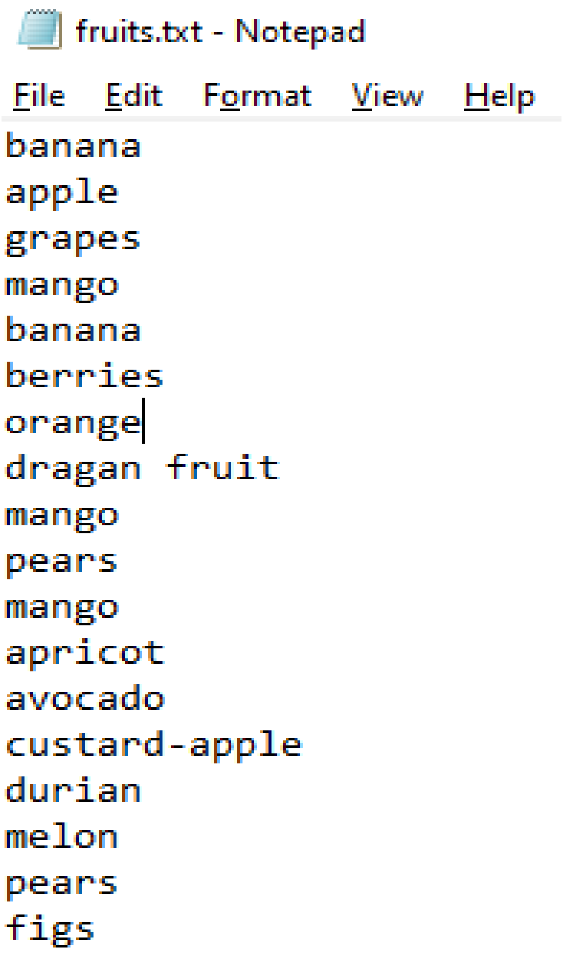
Update

Remove

Shell

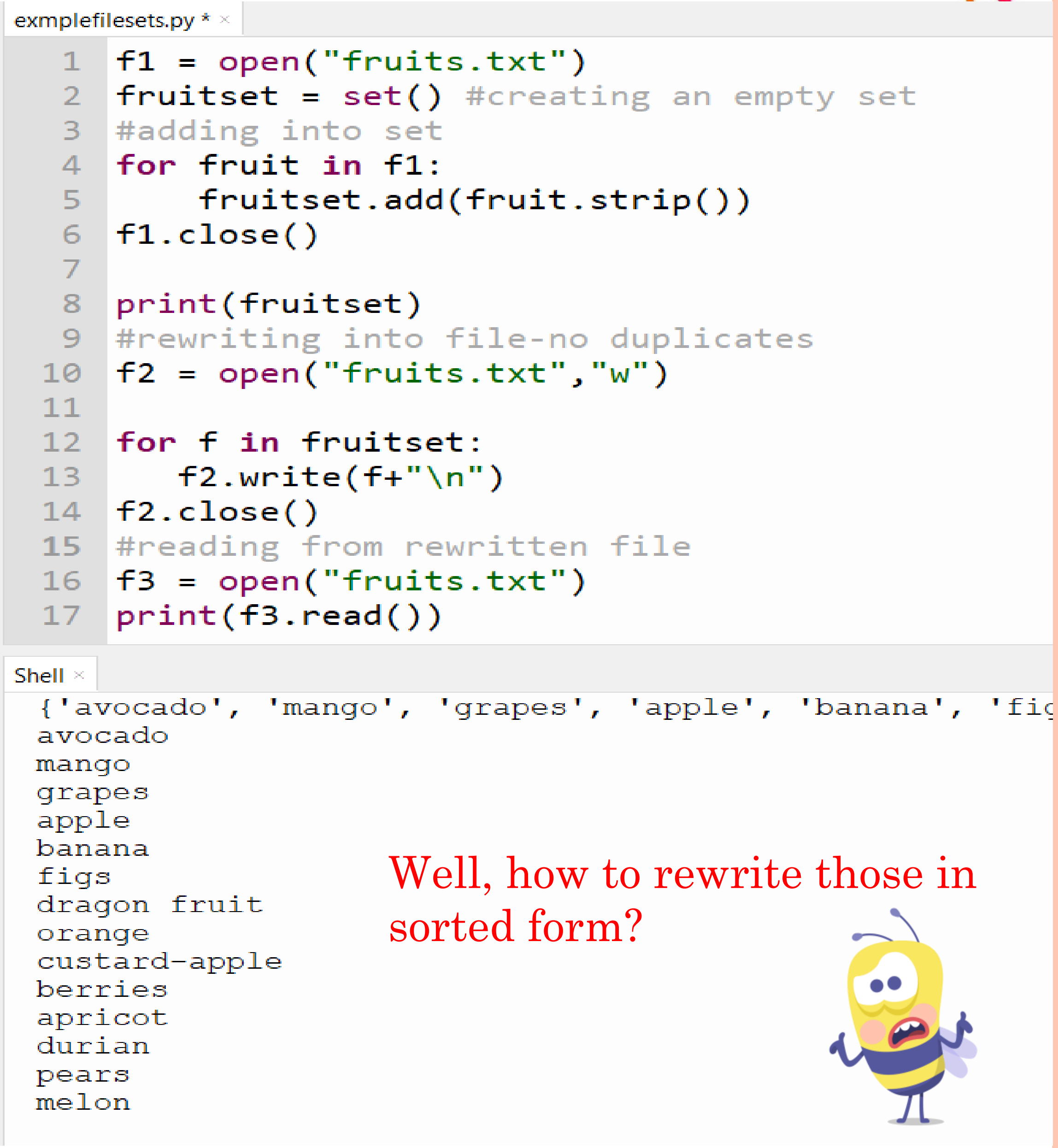
```
Python 3.7.9 (bundled)
>>> %cd 'Z:\Python 2021_Fall\Fall 2021_CT60A0203\Week 8'
>>> %Run example9_set.py

 {'Ashok', 'Kamal', 'Liu', 'Zhao', 'Chen', 'Ren', 'Ajay', 'Andrei', 'Lev', 'Wu'}
 **************
 {'Ashok', 'Kamal', 'Liu', 'Wang', 'Zhao', 'Shu', 'Chen', 'Gao', 'Ren', 'Ajay', 'Andrei', 'Lev', 'Wu'}
 **************
 Enter the name to be removed from Quiz1:Gao
 New list after removal: {'Ashok', 'Kamal', 'Liu', 'Wang', 'Zhao', 'Shu', 'Chen', 'Ren', 'Ajay', 'Andrei', 'Lev', 'Wu'}
```

Example: Suppose the file contain duplicate data which should be removed. How to do that?



fruits.txt - Notepad

```
banana
apple
grapes
mango
banana
berries
orange
dragan fruit
mango
pears
mango
apricot
avocado
custard-apple
durian
melon
pears
figs
```



exmplefilesets.py *

```python
f1 = open("fruits.txt")
fruitset = set() #creating an empty set
#adding into set
for fruit in f1:
     fruitset.add(fruit.strip())
f1.close()

print(fruitset)
#rewriting into file-no duplicates
f2 = open("fruits.txt","w")

for f in fruitset:
    f2.write(f+"\n")
f2.close()
#reading from rewritten file
f3 = open("fruits.txt")
print(f3.read())
```

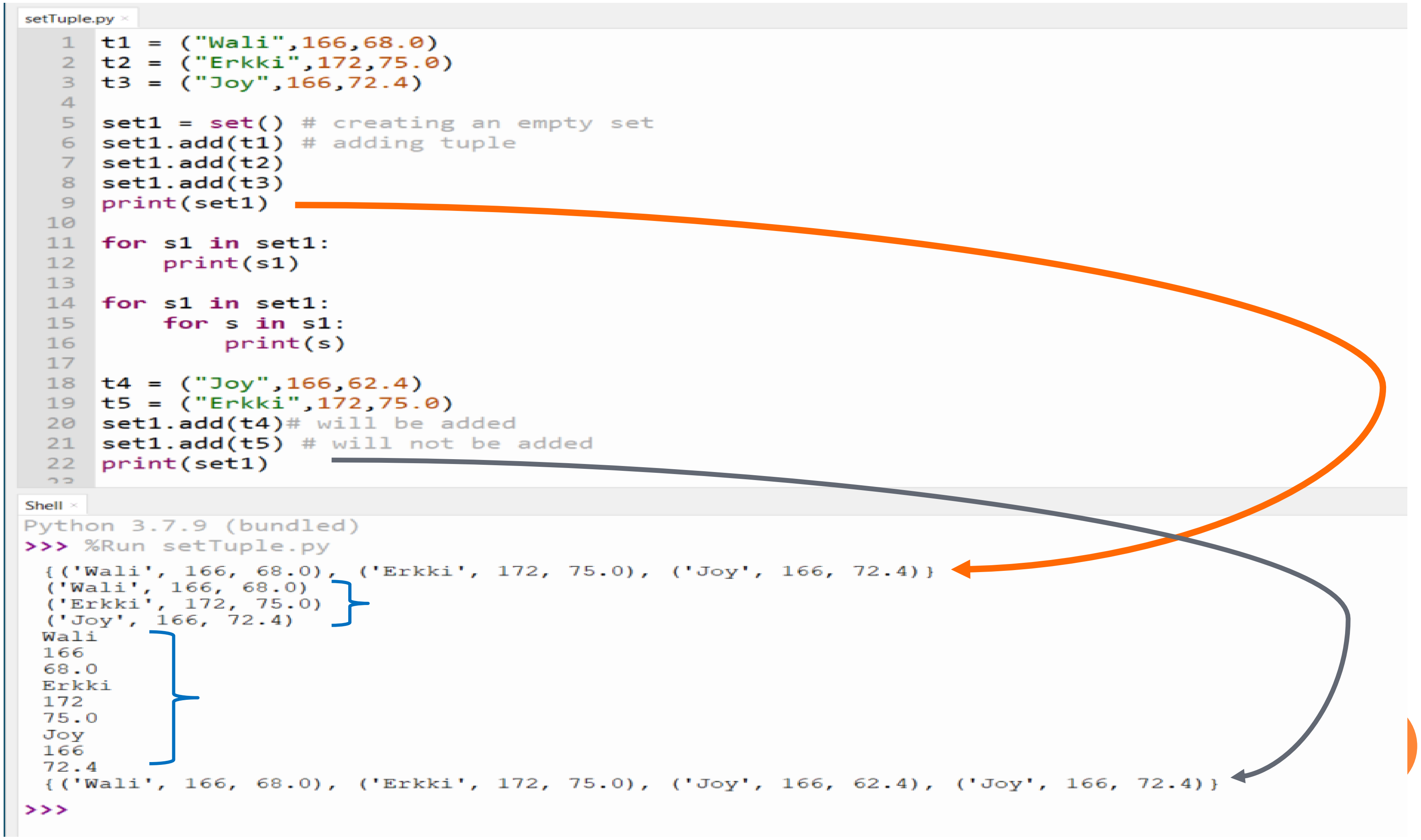
Shell

```
{'avocado', 'mango', 'grapes', 'apple', 'banana', 'fig
avocado
mango
grapes
apple
banana
figs
dragon fruit
orange
custard-apple
berries
apricot
durian
pears
melon
```

Well, how to rewrite those in sorted form?

Is it possible to include one data structure inside of another?

Set can have tuples which is immutable, but it cannot have list and dictionary as they can be mutable. Similarly Lists can have tuple but it can not have sets.



setTuple.py

```python
t1 = ("Wali",166,68.0)
t2 = ("Erkki",172,75.0)
t3 = ("Joy",166,72.4)

set1 = set() # creating an empty set
set1.add(t1) # adding tuple
set1.add(t2)
set1.add(t3)
print(set1)

for s1 in set1:
    print(s1)

for s1 in set1:
    for s in s1:
        print(s)

t4 = ("Joy",166,62.4)
t5 = ("Erkki",172,75.0)
set1.add(t4)# will be added
set1.add(t5) # will not be added
print(set1)
```

Shell

```
Python 3.7.9 (bundled)
>>> %Run setTuple.py
 {('Wali', 166, 68.0), ('Erkki', 172, 75.0), ('Joy', 166, 72.4)}
 ('Wali', 166, 68.0)
 ('Erkki', 172, 75.0)
 ('Joy', 166, 72.4)
 Wali
 166
 68.0
 Erkki
 172
 75.0
 Joy
 166
 72.4
 {('Wali', 166, 68.0), ('Erkki', 172, 75.0), ('Joy', 166, 62.4), ('Joy', 166, 72.4)}
>>>
```

List [ ]	**Tuple ( )**	**Set { }**	**Dictionary { }**
Collection of **ordered** data which can be non-homogeneous data structure	Collection of **ordered** data which can be non-homogeneous data structure	Collection of **unordered** data which can be non-homogeneous data structure	Collection of **ordered** data [key and value] which can be non-homogeneous data structure but key value pair

Example:
L1 = [1, 2, 3, 4]
L2 = [3.4, -9.0, 0.23]
L3 =["A","12B","z123"]
L4 =["A",True,12.4,-90, "789"]
	T1 = (1, 2, 3, 4)
	T2 = (3.4, -9.0, 0.23)
	T3 =("A","12B","z123“)
	T4 =("A",True,12.4,-90, "789“)
		S1 = {1, 2, 3, 4}
		S2 = {3.4, -9.0, 0.23}
		S3 ={"A","12B","z123“}
		S4 ={"A",True,12.4,-90, "789“}
			D1 = {"Name" : "Cho",
			"Height" : 168.0,
			"Age" : 42,
			"Email" :
			"Ch20@gmail.com",
			"Status" : True}

Can have **duplicate** elements	Can have **duplicate** elements	**Can not** have duplicate elements	**Can not** have duplicate **keys** but values
To create an empty list - Example **List1 = [ ]**	To create an empty tuple - Example **Tuple1 = tuple( )**	To create an empty set - Example **Set1 = set( )**	To create an empty dictionary - Example **Dict1 = { }**
Mutable, so elements in the list can be changed	**Immutable, so elements in the tuple can not be changed**	The elements of the set are immutable, that is, they cannot be changed.	Mutable. But Keys are not duplicated