Sets

- Set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable.
- However, the set itself is mutable (we can add or remove items).
- Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.
- A set is a dictionary with no values. It has only unique keys. It is like a dictionary. But we omit the values, and can use complex, powerful set logic.

Sets

- sets are less often used. The initializer syntax is the same as a dictionary, but no pairs are specified—just keys.
- A frozenset is immutable and can be a dictionary key.

Creation

- This program initializes a set. When we initialize a set, we do not include the values in the syntax as with a dictionary. We specify only the keys.

Python program that creates set

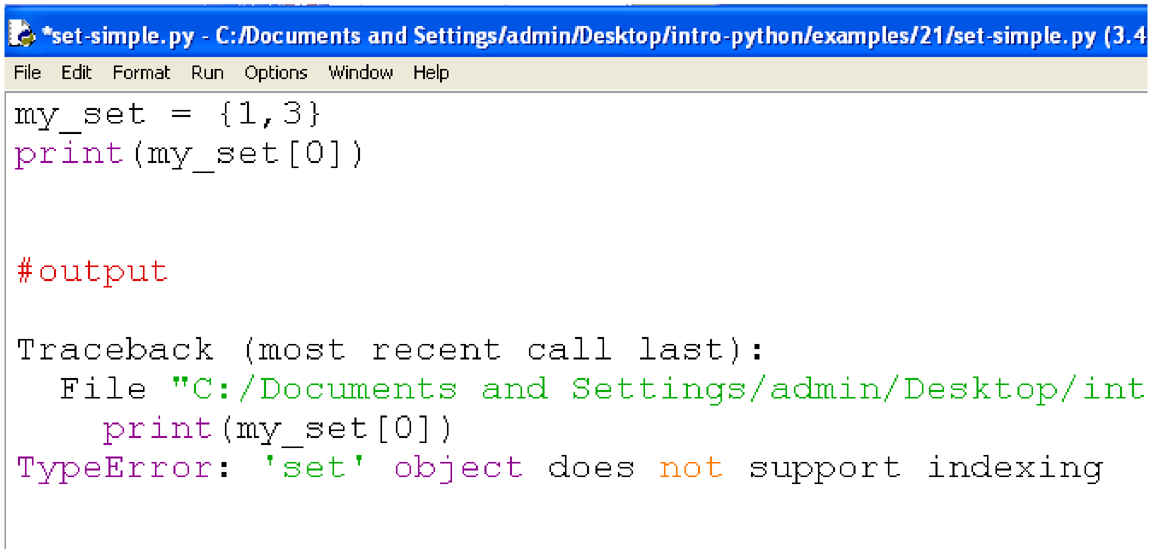
```
# Create a set.
items = {"arrow", "spear", "arrow", "arrow", "rock"}

# Print set.
print(items)
```

Output

```
{'spear', 'arrow', 'rock'}
```

Access

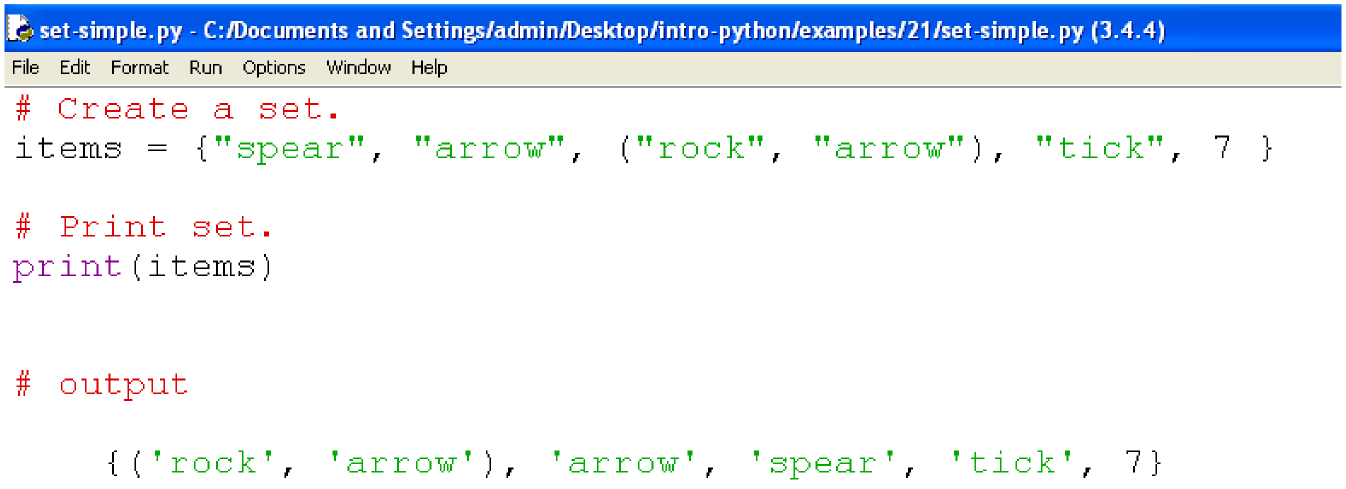
- Sets are mutable. But since they are unordered, indexing have no meaning. We cannot access or change an element of set using indexing or slicing. Set does not support it. We can add single elements using the method add(). Multiple elements can be added using update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.


*set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4
File Edit Format Run Options Window Help

```
my_set = {1,3}
print(my_set[0])


#output

Traceback (most recent call last):
  File "C:/Documents and Settings/admin/Desktop/int
    print(my_set[0])
TypeError: 'set' object does not support indexing
```

Nesting

- set can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
# Create a set.
items = {"spear", "arrow", ("rock", "arrow"), "tick", 7 }

# Print set.
print(items)


# output

    {('rock', 'arrow'), 'arrow', 'spear', 'tick', 7}
```

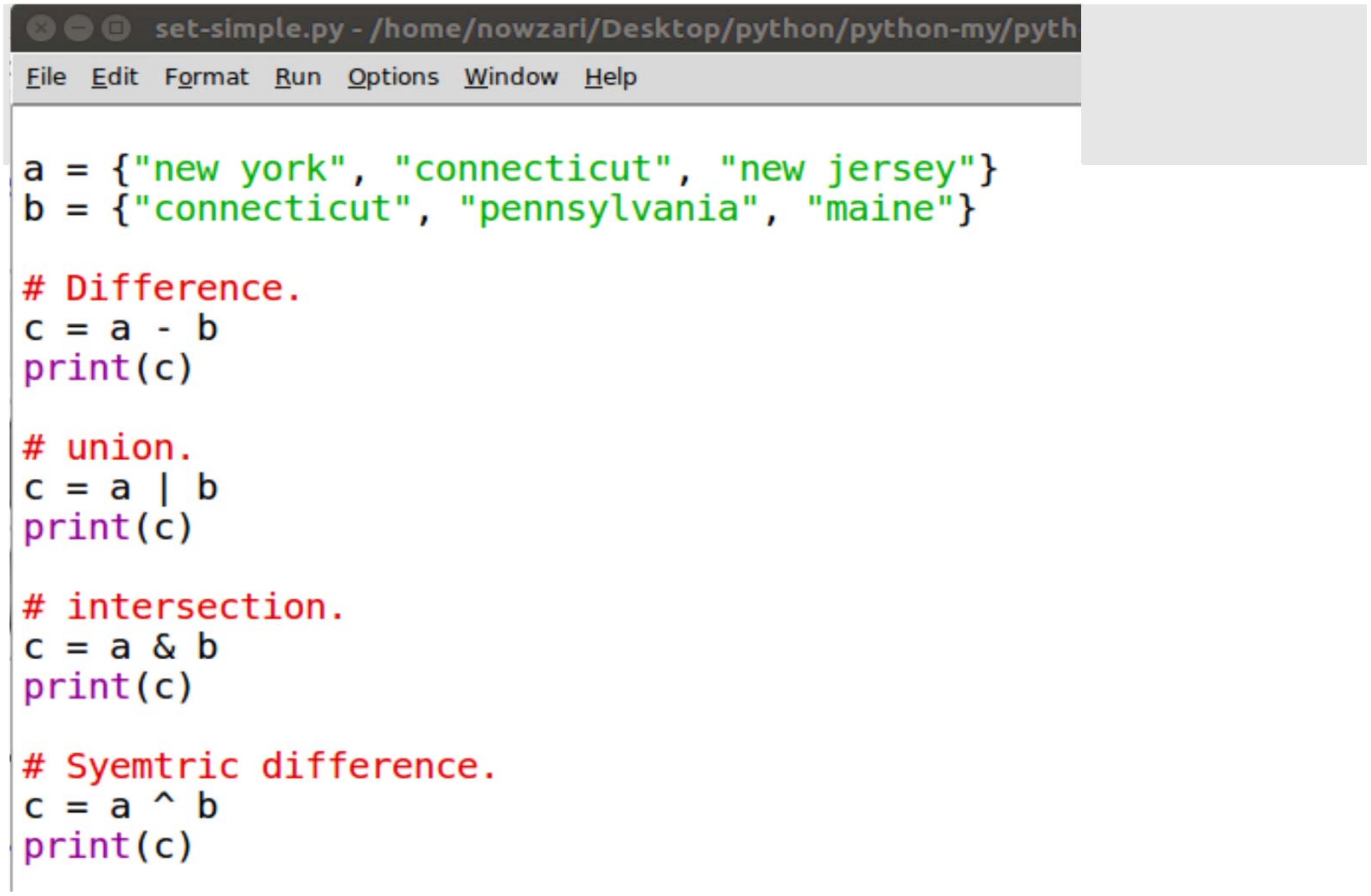
Nesting

- set cannot have mutable items

- **my_set = {1, 2, [3, 4]}**
- **TypeError: unhashable type: 'list'**

Operators for set

- k in d : This returns true or false depending on whether the item exists.
- k not in d: True, if a value k doesn't exist in the set d
- Union (|), intersection (&) difference (-), symmetric difference(^)

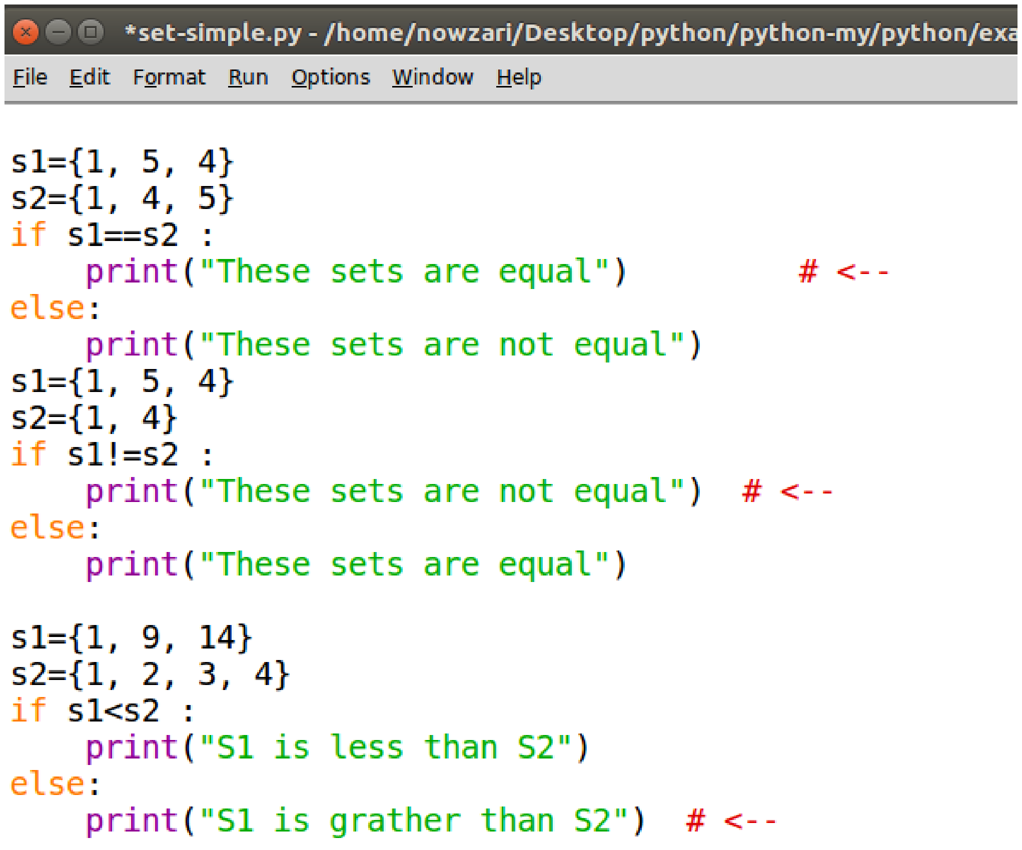

set-simple.py - /home/nowzari/Desktop/python/python-my/pyth
File Edit Format Run Options Window Help

```python
a = {"new york", "connecticut", "new jersey"}
b = {"connecticut", "pennsylvania", "maine"}

# Difference.
c = a - b
print(c)

# union.
c = a | b
print(c)

# intersection.
c = a & b
print(c)

# Syemtric difference.
c = a ^ b
print(c)
```

```
{'new jersey', 'new york'}
{'pennsylvania', 'maine', 'new jersey', 'new york', 'connecticut'}
{'connecticut'}
{'pennsylvania', 'maine', 'new jersey', 'new york'}
```

Set Comparison

- The operators >, <, == , !=, <=. >= compares elements of two sets.


*set-simple.py - /home/nowzari/Desktop/python/python-my/python/exa
File Edit Format Run Options Window Help

```
s1={1, 5, 4}
s2={1, 4, 5}
if s1==s2 :
    print("These sets are equal")         # <--
else:
    print("These sets are not equal")
s1={1, 5, 4}
s2={1, 4}
if s1!=s2 :
    print("These sets are not equal")  # <--
else:
    print("These sets are equal")

s1={1, 9, 14}
s2={1, 2, 3, 4}
if s1<s2 :
    print("S1 is less than S2")
else:
    print("S1 is grather than S2")  # <--
```

Built-in set Functions

- Python includes the following set functions

- len(d): returns the number of entries,
- type(d): Return the type set

Len, in

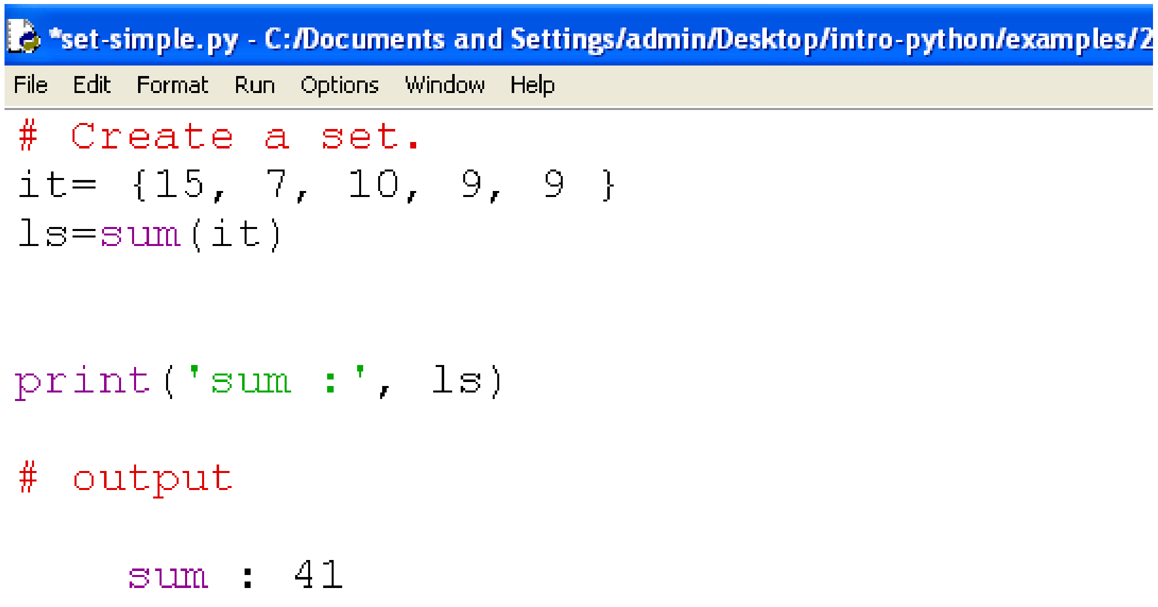
Python program that creates set

```
# Create a set.
items = {"arrow", "spear", "arrow", "arrow", "rock"}

# Print set.
print(items)
print(len(items))

# Use in-keyword.
if "rock" in items:
    print("Rock exists")

# Use not-in keywords.
if "clock" not in items:
    print("Cloak not found")
```

Output

```
{'spear', 'arrow', 'rock'}
3
Rock exists
Cloak not found
```

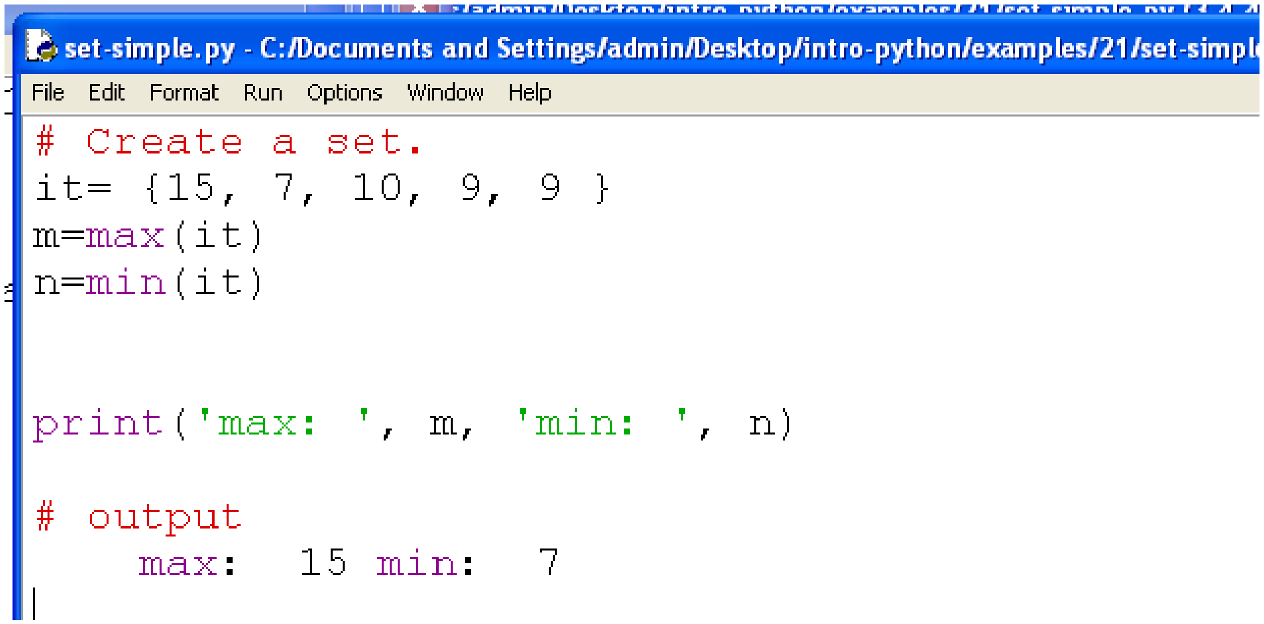
Sum


*set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/2
File Edit Format Run Options Window Help

```
# Create a set.
it= {15, 7, 10, 9, 9 }
ls=sum(it)


print('sum :', ls)

# output

    sum : 41
```

Max and min

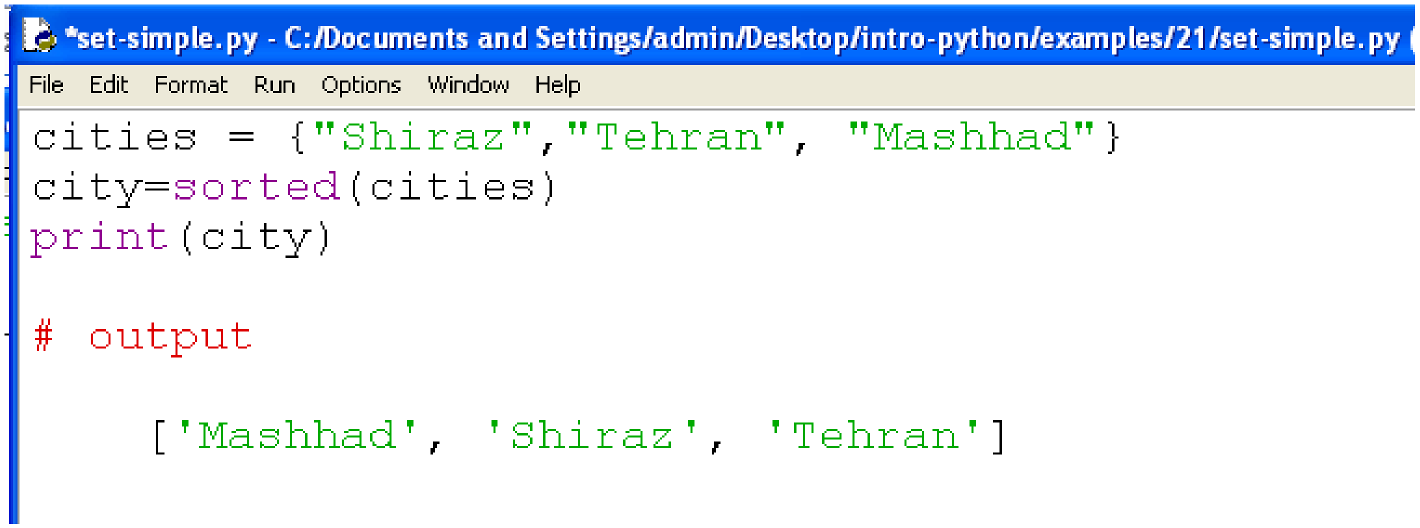
- Return the largest (smallest) item in the set.


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simpl
File Edit Format Run Options Window Help

```
# Create a set.
it= {15, 7, 10, 9, 9 }
m=max(it)
n=min(it)


print('max: ', m, 'min: ', n)

# output
    max:  15 min:  7
```

Sorted

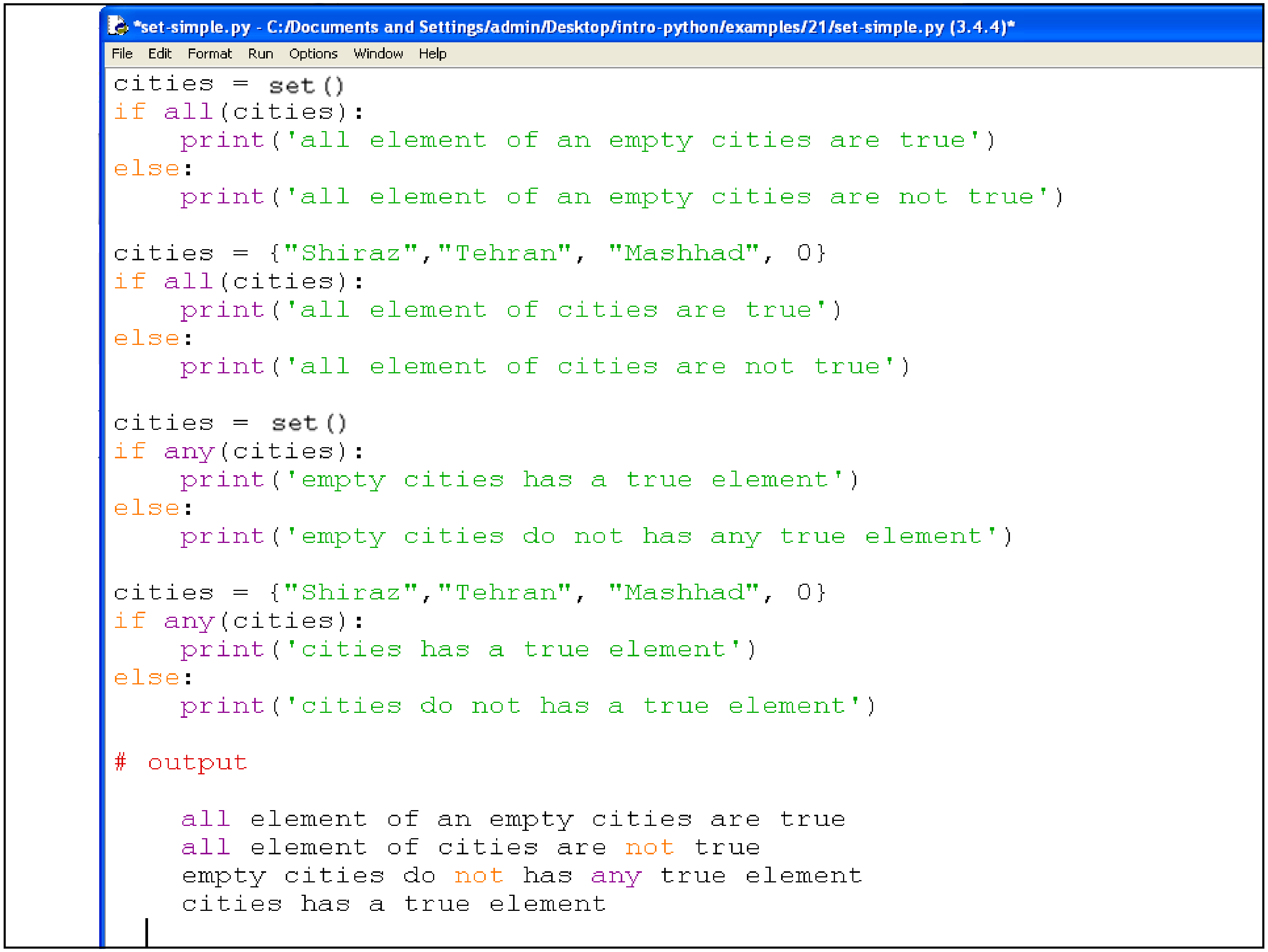
- Return a new sorted list from elements in the set(does not sort the set itself).


*set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py
File Edit Format Run Options Window Help

```
cities = {"Shiraz","Tehran", "Mashhad"}
city=sorted(cities)
print(city)

# output

     ['Mashhad', 'Shiraz', 'Tehran']
```

All, Any

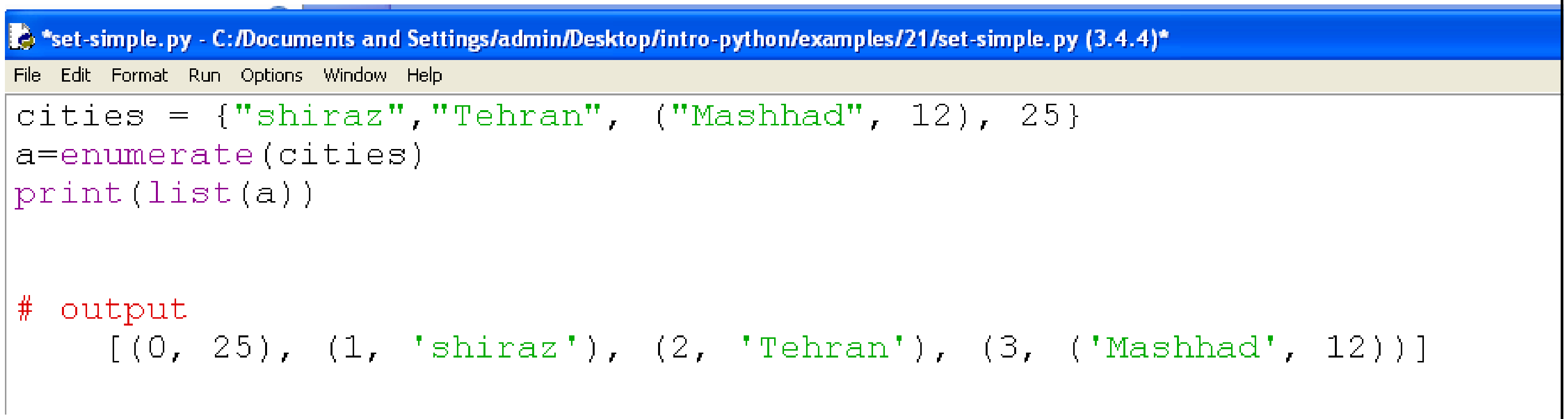
- All: Return True if all elements of the set are true (or if the set is empty).
- Any: Return True if any element of the set is true. If the set is empty, return False.


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
cities = set()
if all(cities):
    print('all element of an empty cities are true')
else:
    print('all element of an empty cities are not true')

cities = {"Shiraz","Tehran", "Mashhad", 0}
if all(cities):
    print('all element of cities are true')
else:
    print('all element of cities are not true')

cities = set()
if any(cities):
    print('empty cities has a true element')
else:
    print('empty cities do not has any true element')

cities = {"Shiraz","Tehran", "Mashhad", 0}
if any(cities):
    print('cities has a true element')
else:
    print('cities do not has a true element')

# output

    all element of an empty cities are true
    all element of cities are not true
    empty cities do not has any true element
    cities has a true element
```

Enumerate

- Return an enumerate object. It contains the index and value of all the items of set as a pair.


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
cities = {"shiraz","Tehran", ("Mashhad", 12), 25}
a=enumerate(cities)
print(list(a))


# output
    [(0, 25), (1, 'shiraz'), (2, 'Tehran'), (3, ('Mashhad', 12))]
```

Set Type conversion

- We can convert some object to set and vice versa

set

- We can create a set from a list (or other iterable collection). Consider this example. We pass a list with six different elements in it to set(). The duplicates are ignored.

Python program that creates set from list

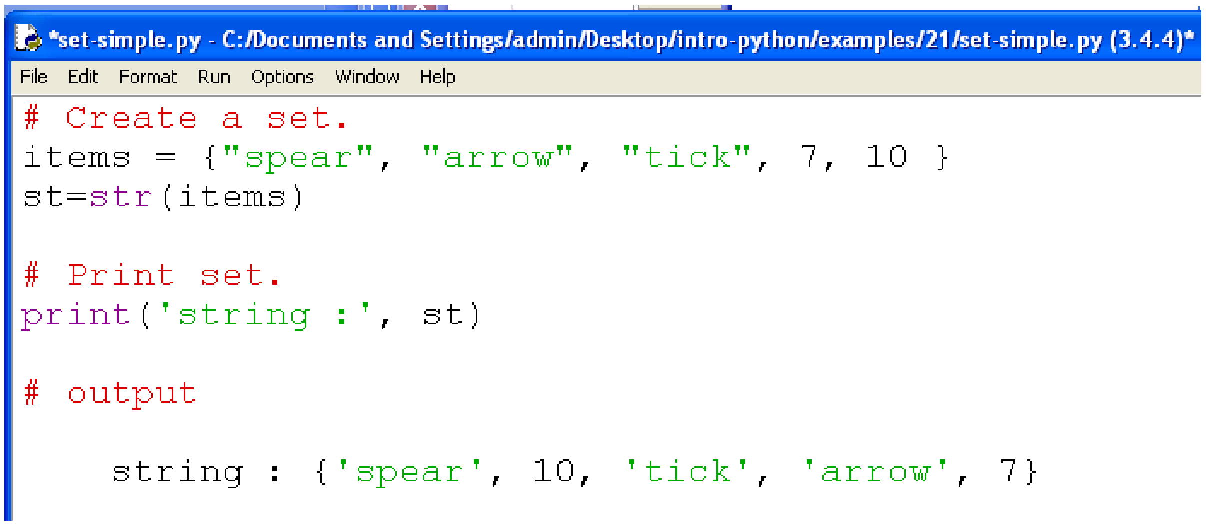
```
# Create a set from this list.
# ... Duplicates are ignored.
numbers = set([10, 20, 20, 30, 40, 50])
print(numbers)
```

Output

```
{40, 10, 20, 50, 30}
```

Str

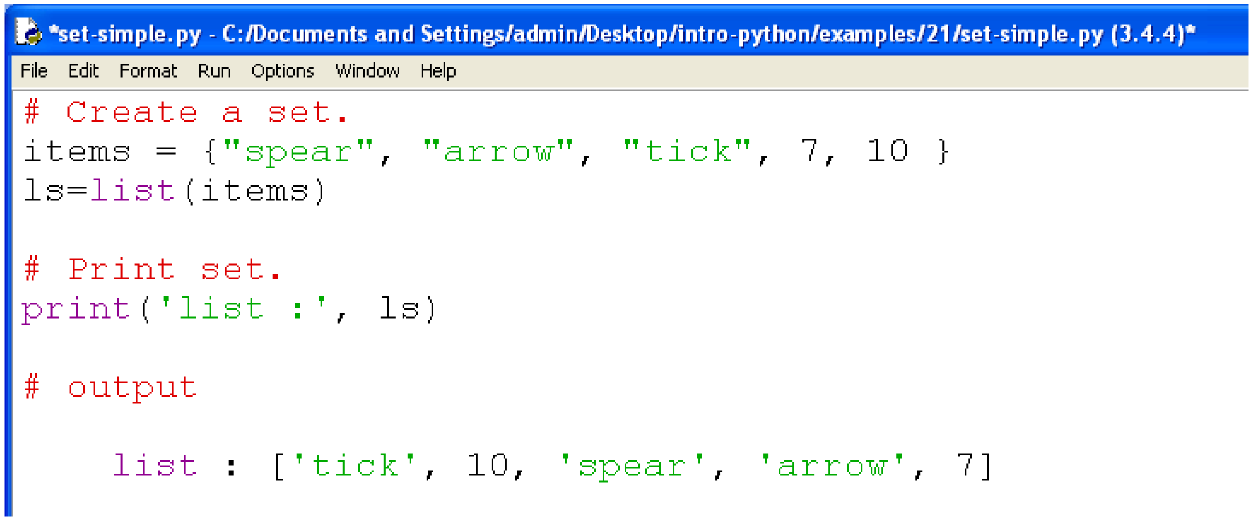
- We can create a string from a set


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
# Create a set.
items = {"spear", "arrow", "tick", 7, 10 }
st=str(items)

# Print set.
print('string :', st)

# output

    string : {'spear', 10, 'tick', 'arrow', 7}
```

List

- We can create a list from a set

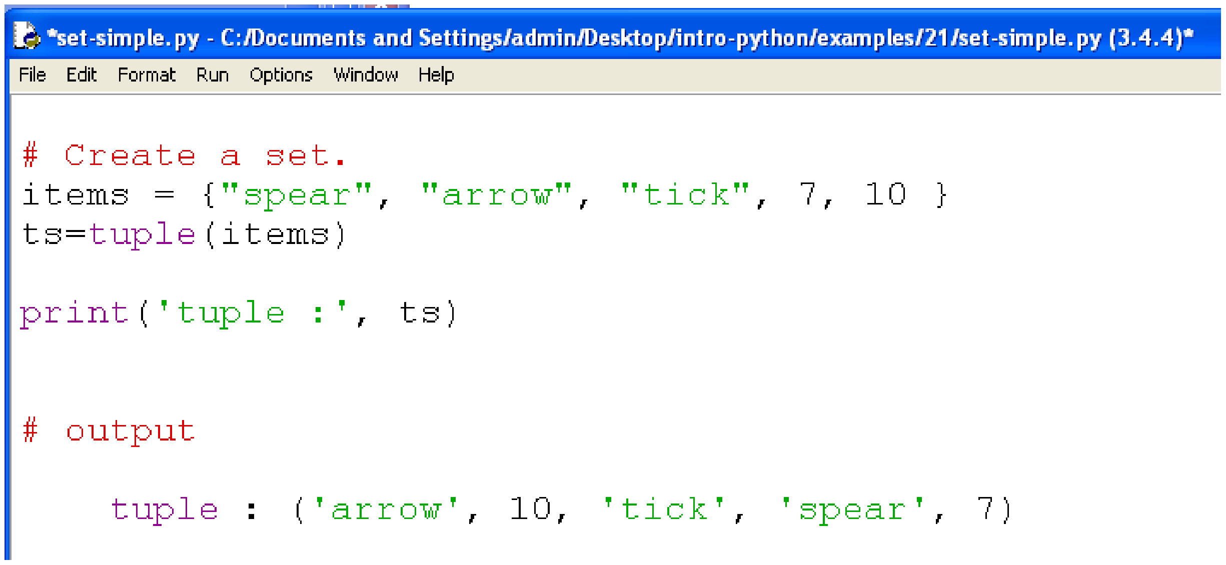


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
# Create a set.
items = {"spear", "arrow", "tick", 7, 10 }
ls=list(items)

# Print set.
print('list :', ls)

# output

    list : ['tick', 10, 'spear', 'arrow', 7]
```

Tuple

- We can create a tuple from a set


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
# Create a set.
items = {"spear", "arrow", "tick", 7, 10 }
ts=tuple(items)

print('tuple :', ts)


# output

    tuple : ('arrow', 10, 'tick', 'spear', 7)
```

Type specific functions for sets

- There are some type specific functions for **sets**, such as: clear, copy.

Add

- Elements can be added to a set with the add method. Here we create an empty set with the set built-in method. We then add three elements to the set.

Python program that adds, discards, removes

```
# An empty set.
items = set()

# Add three strings.
items.add("cat")
items.add("dog")
items.add("gerbil")
print(items)
```

Output

```
{'gerbil', 'dog', 'cat'}
```

Subset, superset

- In set theory, we determine relations between sets of elements. And with the Python set type, we can compute these with built-in methods.
- Is subset: This returns true if a set is a subset of other set.
- Is superset:This method also returns true if a set is a superset of other set.

Intersection

- This method returns a new set that contains just the shared numbers. Other values are omitted.

example

Python program that uses set methods

```
numbers1 = {1, 3, 5, 7}
numbers2 = {1, 3}

# Is subset.
if numbers2.issubset(numbers1):
    print("Is a subset")

# Is superset.
if numbers1.issuperset(numbers2):
    print("Is a superset")

# Intersection of the two sets.
print(numbers1.intersection(numbers2))
```

Output

```
Is a subset
Is a superset
{1, 3}
```

Union

- Another set operation that is available is union(). This combines two sets. Any element in either set is retained in the return value of union. But duplicates are eliminated.

Python program that unions two sets

```
# Two sets.
set1 = {1, 2, 3}
set2 = {6, 5, 4, 3}

# Union the sets.
set3 = set1.union(set2)
print(set3)
```

Output

```
{1, 2, 3, 4, 5, 6}
```

Subtract, difference

- A set can be subtracted from another set. The difference() method is used in this case. The syntax that is clearer is the best choice.

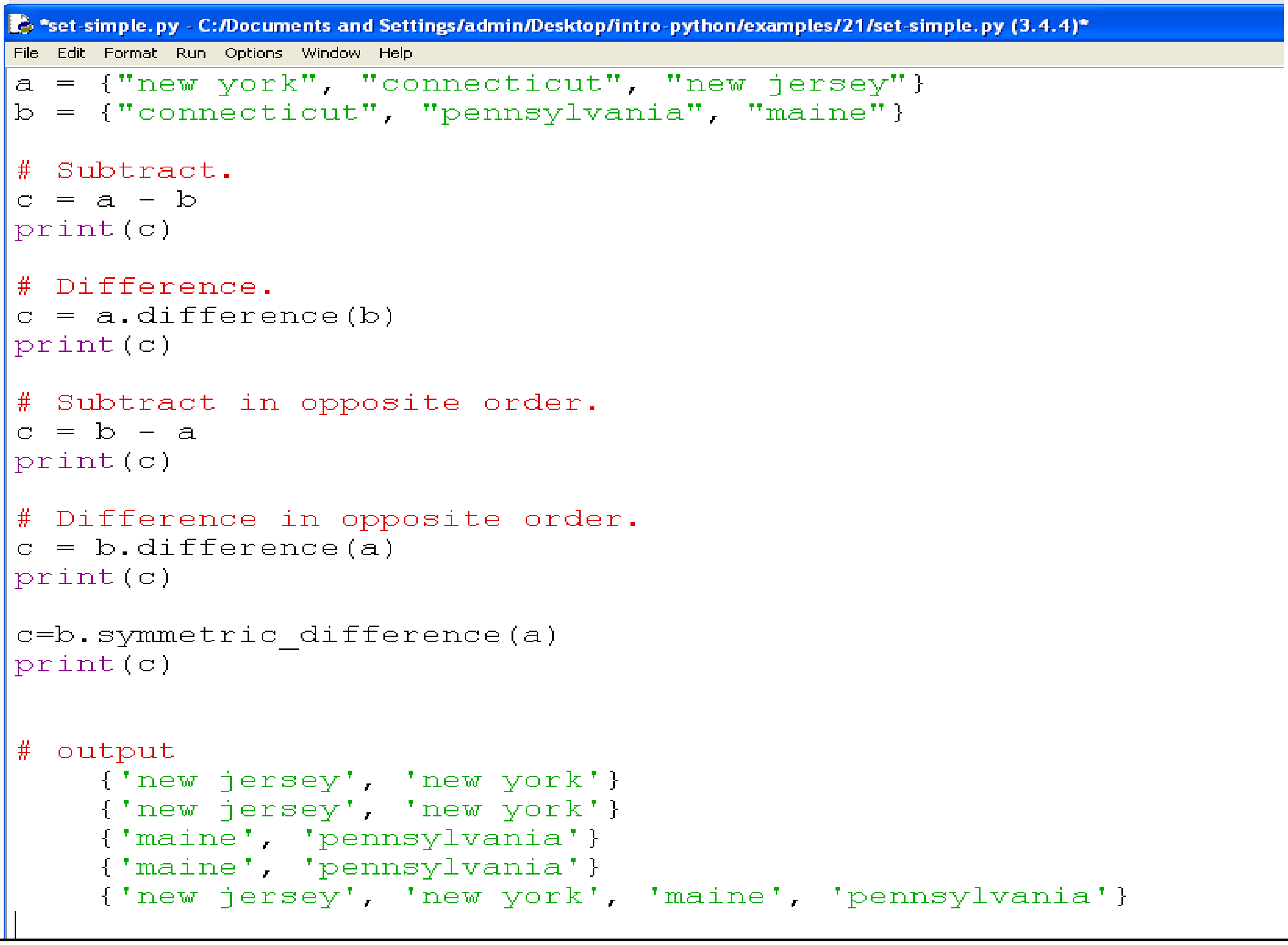
Python program that subtracts sets

```
a = {"new york", "connecticut", "new jersey"}
b = {"connecticut", "pennsylvania", "maine"}

# Subtract.
c = a - b
print(c)

# Difference.
c = a.difference(b)
print(c)

# Subtract in opposite order.
c = b - a
print(c)

# Difference in opposite order.
c = b.difference(a)
print(c)
```

Output

```
{'new jersey', 'new york'}
{'new jersey', 'new york'}
{'pennsylvania', 'maine'}
{'pennsylvania', 'maine'}
```

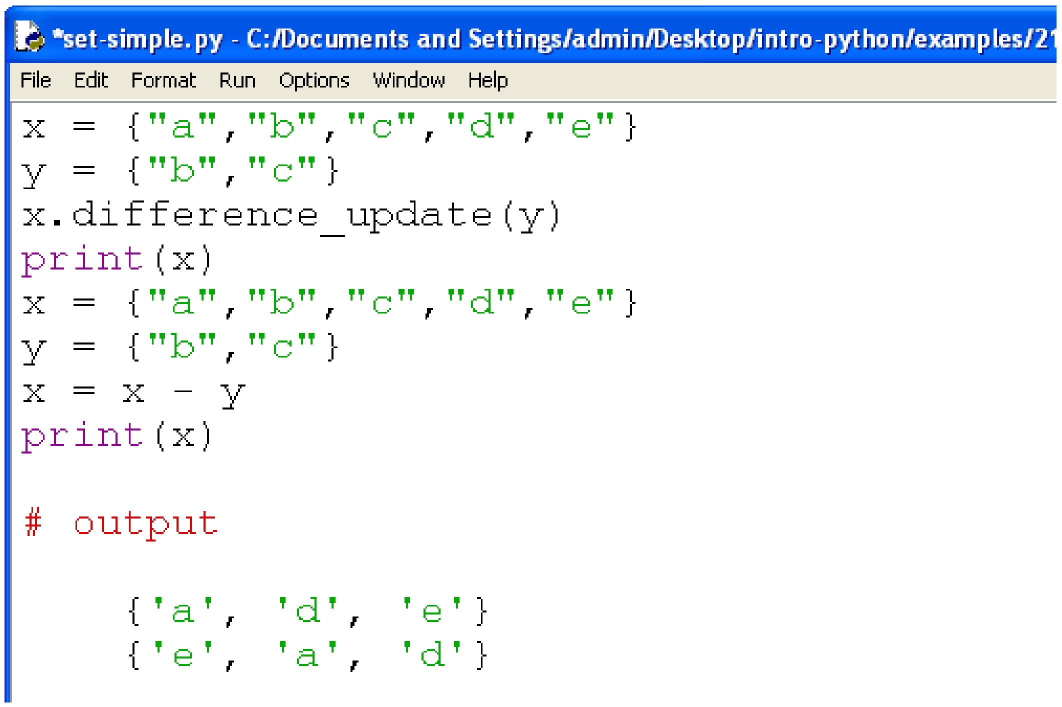
Symmetric difference


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
a = {"new york", "connecticut", "new jersey"}
b = {"connecticut", "pennsylvania", "maine"}

# Subtract.
c = a - b
print(c)

# Difference.
c = a.difference(b)
print(c)

# Subtract in opposite order.
c = b - a
print(c)

# Difference in opposite order.
c = b.difference(a)
print(c)

c=b.symmetric_difference(a)
print(c)


# output
    {'new jersey', 'new york'}
    {'new jersey', 'new york'}
    {'maine', 'pennsylvania'}
    {'maine', 'pennsylvania'}
    {'new jersey', 'new york', 'maine', 'pennsylvania'}
```

Difference update

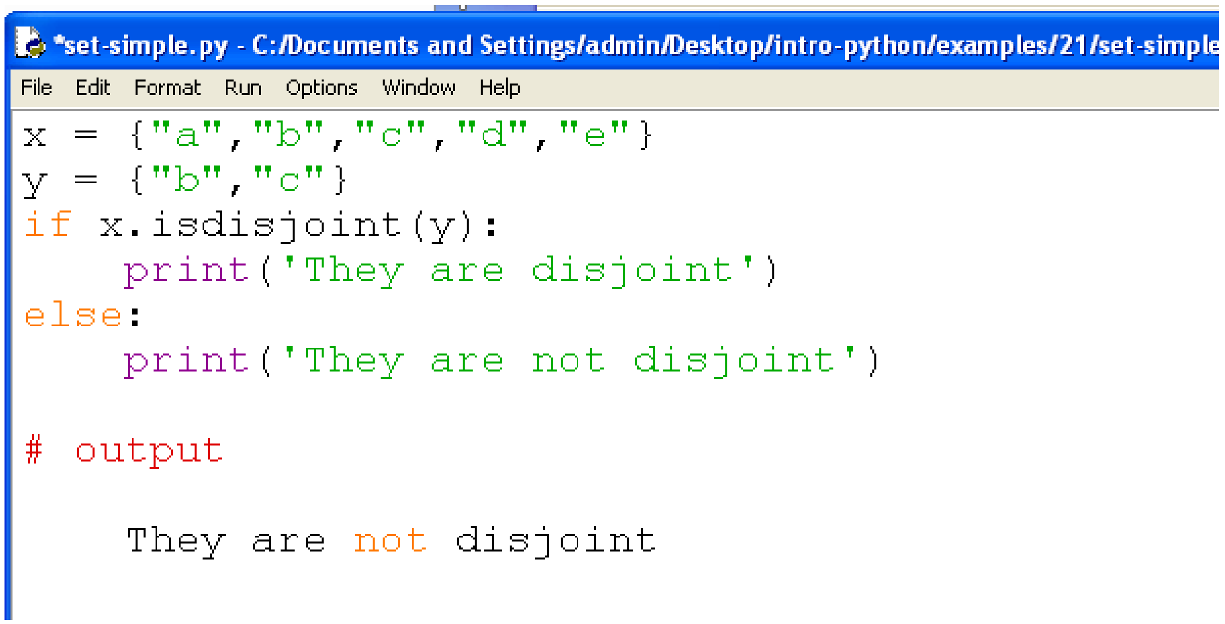
- The method difference_update removes all elements of another set from this set. x.difference_update(y) is the same as "x = x - y"


*set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21
File Edit Format Run Options Window Help

```
x = {"a","b","c","d","e"}
y = {"b","c"}
x.difference_update(y)
print(x)
x = {"a","b","c","d","e"}
y = {"b","c"}
x = x - y
print(x)

# output

    {'a', 'd', 'e'}
    {'e', 'a', 'd'}
```

Isdisjoint

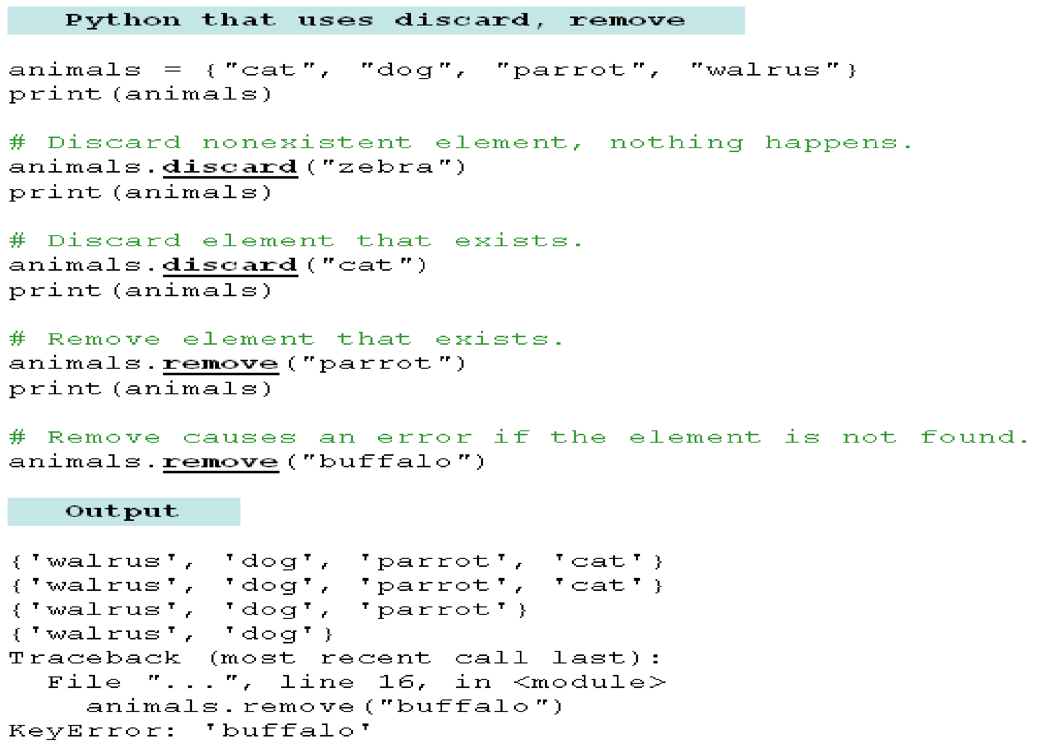
- This method returns True if two sets have a null intersection.


*set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple
File Edit Format Run Options Window Help

```
x = {"a","b","c","d","e"}
y = {"b","c"}
if x.isdisjoint(y):
    print('They are disjoint')
else:
    print('They are not disjoint')

# output

    They are not disjoint
```

Discard, remove

- We pass discard() the value of an element we want to remove. If the element does not exist, discard will cause no error—it does nothing. Remove, however, will cause a KeyError.



Python that uses discard, remove

```
animals = {"cat", "dog", "parrot", "walrus"}
print(animals)

# Discard nonexistent element, nothing happens.
animals.discard("zebra")
print(animals)

# Discard element that exists.
animals.discard("cat")
print(animals)

# Remove element that exists.
animals.remove("parrot")
print(animals)

# Remove causes an error if the element is not found.
animals.remove("buffalo")
```

Output

```
{'walrus', 'dog', 'parrot', 'cat'}
{'walrus', 'dog', 'parrot', 'cat'}
{'walrus', 'dog', 'parrot'}
{'walrus', 'dog'}
Traceback (most recent call last):
  File "...", line 16, in <module>
    animals.remove("buffalo")
KeyError: 'buffalo'
```

Discard, remove

Python that uses discard, remove

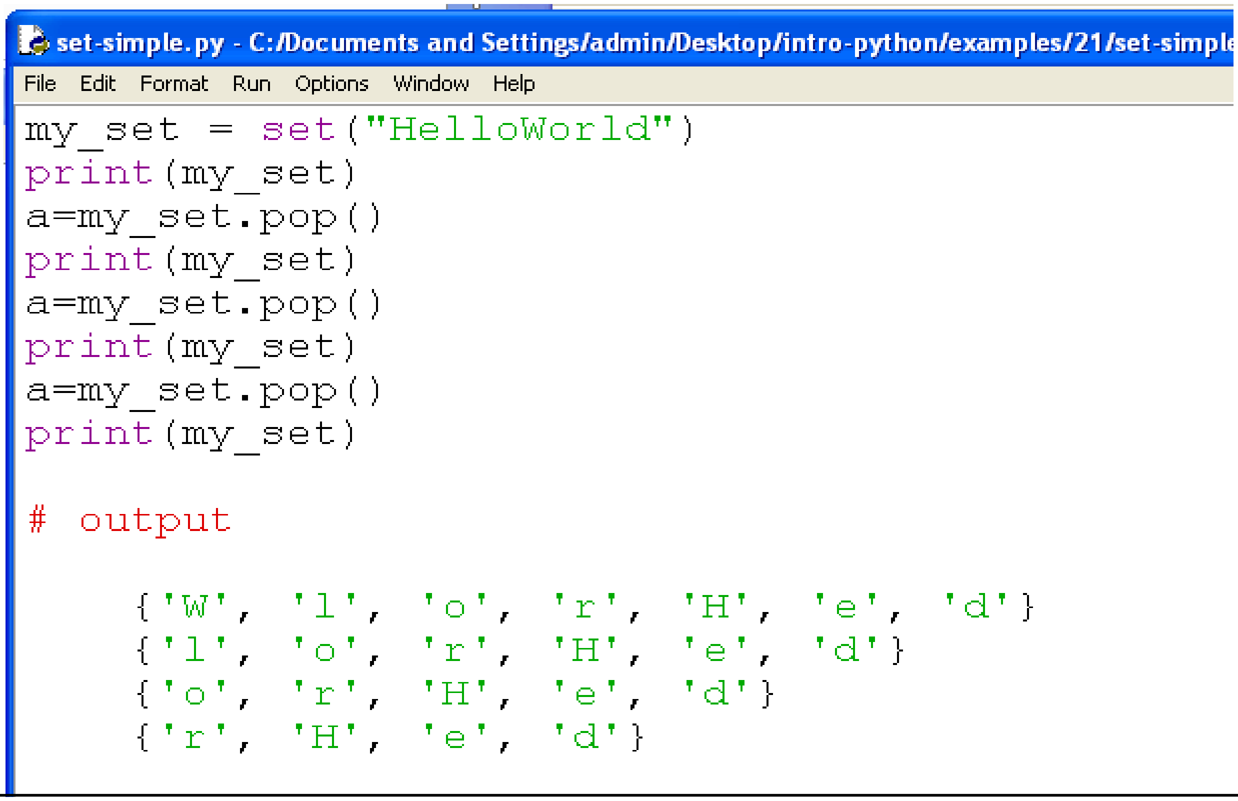
```
animals = {"cat", "dog", "parrot", "walrus"}
print(animals)

# Discard nonexistent element, nothing happens.
animals.discard("zebra")
print(animals)

# Discard element that exists.
animals.discard("cat")
print(animals)

# Remove element that exists.
animals.remove("parrot")
print(animals)

# Remove causes an error if the element is not found.
animals.remove("buffalo")
```

Output

```
{'walrus', 'dog', 'parrot', 'cat'}
{'walrus', 'dog', 'parrot', 'cat'}
{'walrus', 'dog', 'parrot'}
{'walrus', 'dog'}
Traceback (most recent call last):
  File "...", line 16, in <module>
    animals.remove("buffalo")
KeyError: 'buffalo'
```

Pop

- Similarly, we can remove and return an item using the pop() method. Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple
File Edit Format Run Options Window Help

```
my_set = set("HelloWorld")
print(my_set)
a=my_set.pop()
print(my_set)
a=my_set.pop()
print(my_set)
a=my_set.pop()
print(my_set)

# output

    {'W', 'l', 'o', 'r', 'H', 'e', 'd'}
    {'l', 'o', 'r', 'H', 'e', 'd'}
    {'o', 'r', 'H', 'e', 'd'}
    {'r', 'H', 'e', 'd'}
```

Get keys, dictionary

- A dictionary contains only unique keys. With the set() built-in, we can get these keys and convert them into a set.

Python that uses dictionary, set keys

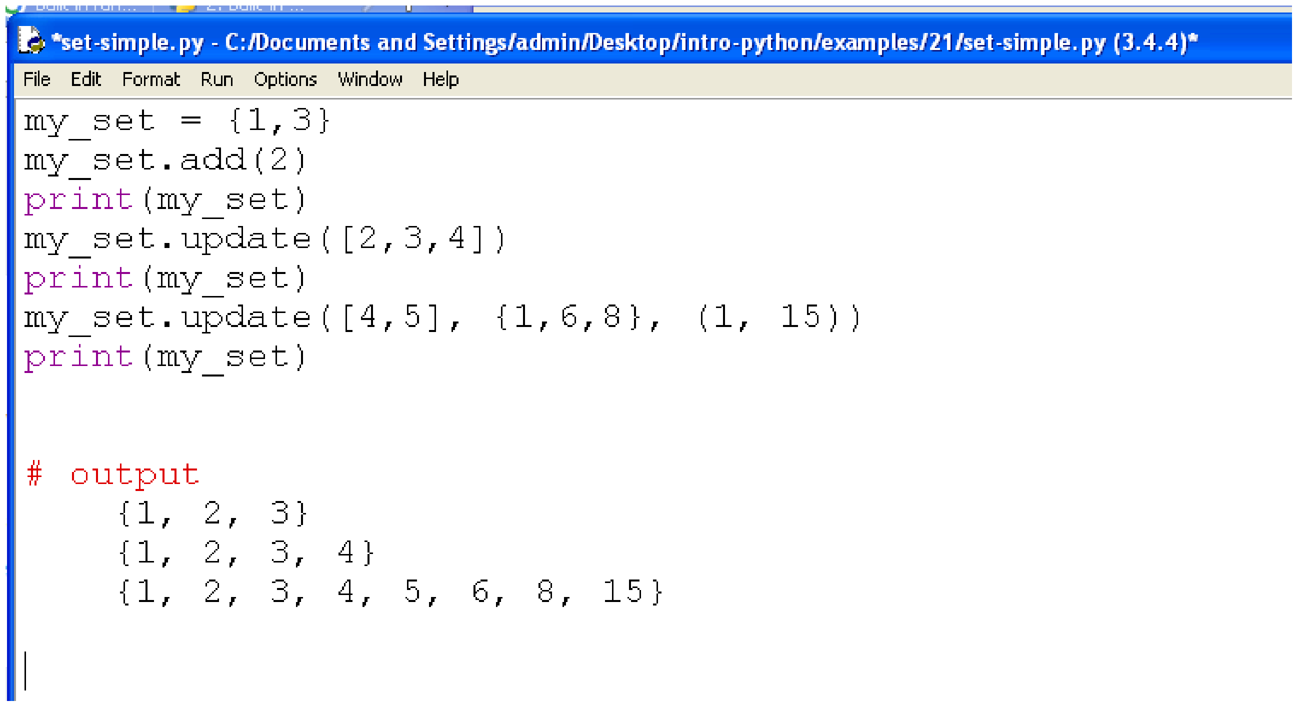
```
# This dictionary contains key-value pairs.
dictionary = {"cat": 1, "dog": 2, "bird": 3}
print(dictionary)

# This set contains just the dictionary's keys.
keys = set(dictionary)
print(keys)
```

Output

```
{'bird': 3, 'dog': 2, 'cat': 1}
{'cat', 'bird', 'dog'}
```

Update

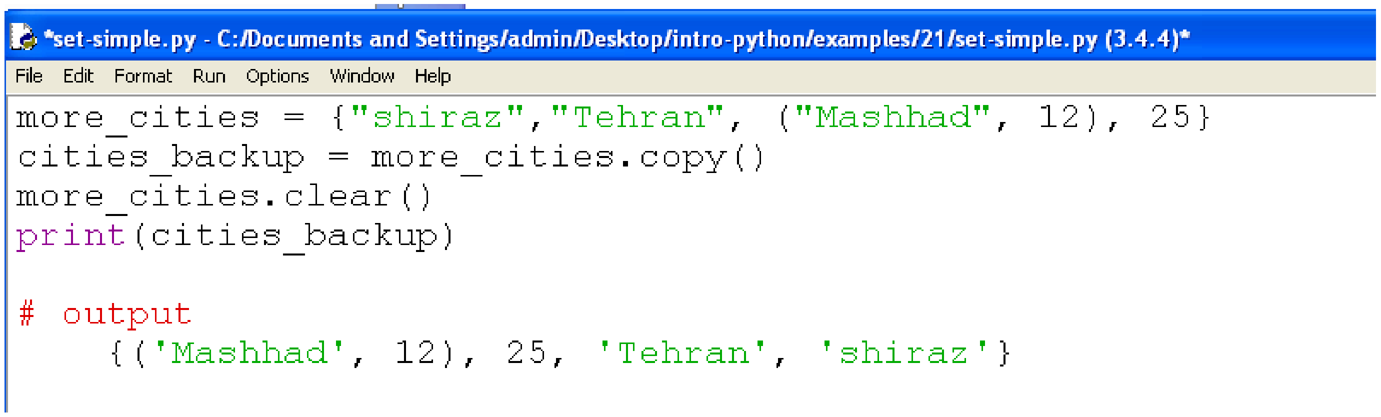
- Multiple elements can be added using update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
my_set = {1,3}
my_set.add(2)
print(my_set)
my_set.update([2,3,4])
print(my_set)
my_set.update([4,5], {1,6,8}, (1, 15))
print(my_set)


#  output
    {1, 2, 3}
    {1, 2, 3, 4}
    {1, 2, 3, 4, 5, 6, 8, 15}
```

Copy

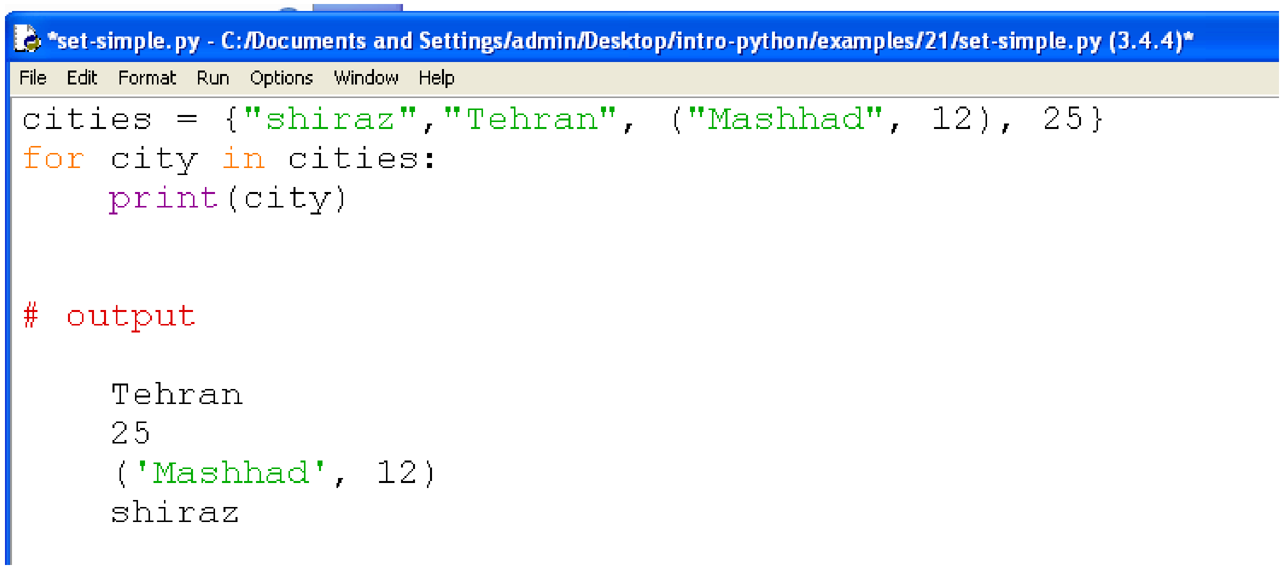
- Return a shallow copy of a set


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)

```
more_cities = {"shiraz","Tehran", ("Mashhad", 12), 25}
cities_backup = more_cities.copy()
more_cities.clear()
print(cities_backup)

# output
    {('Mashhad', 12), 25, 'Tehran', 'shiraz'}
```

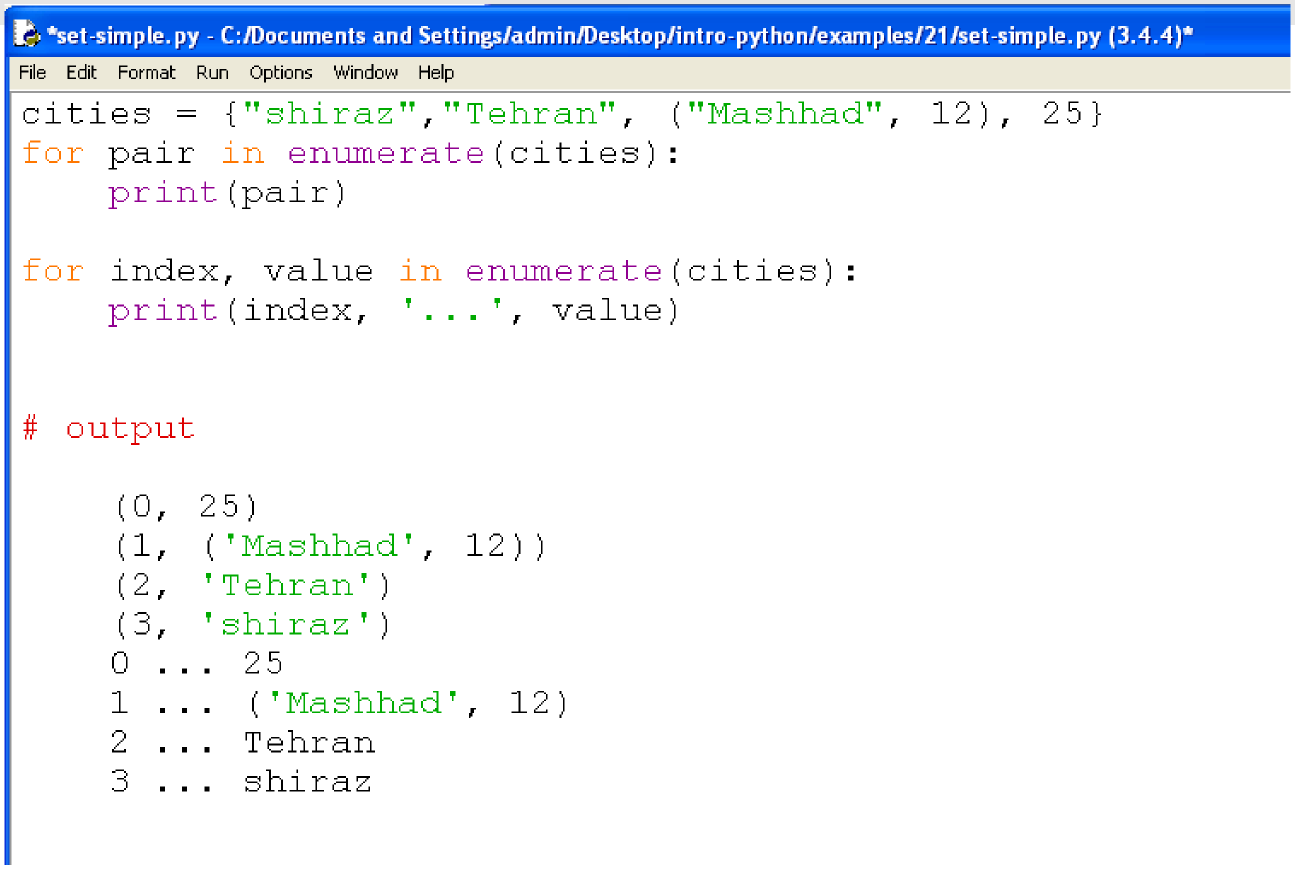
Iterating Through a Set

- Using a for loop we can iterate though each item in a set.


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

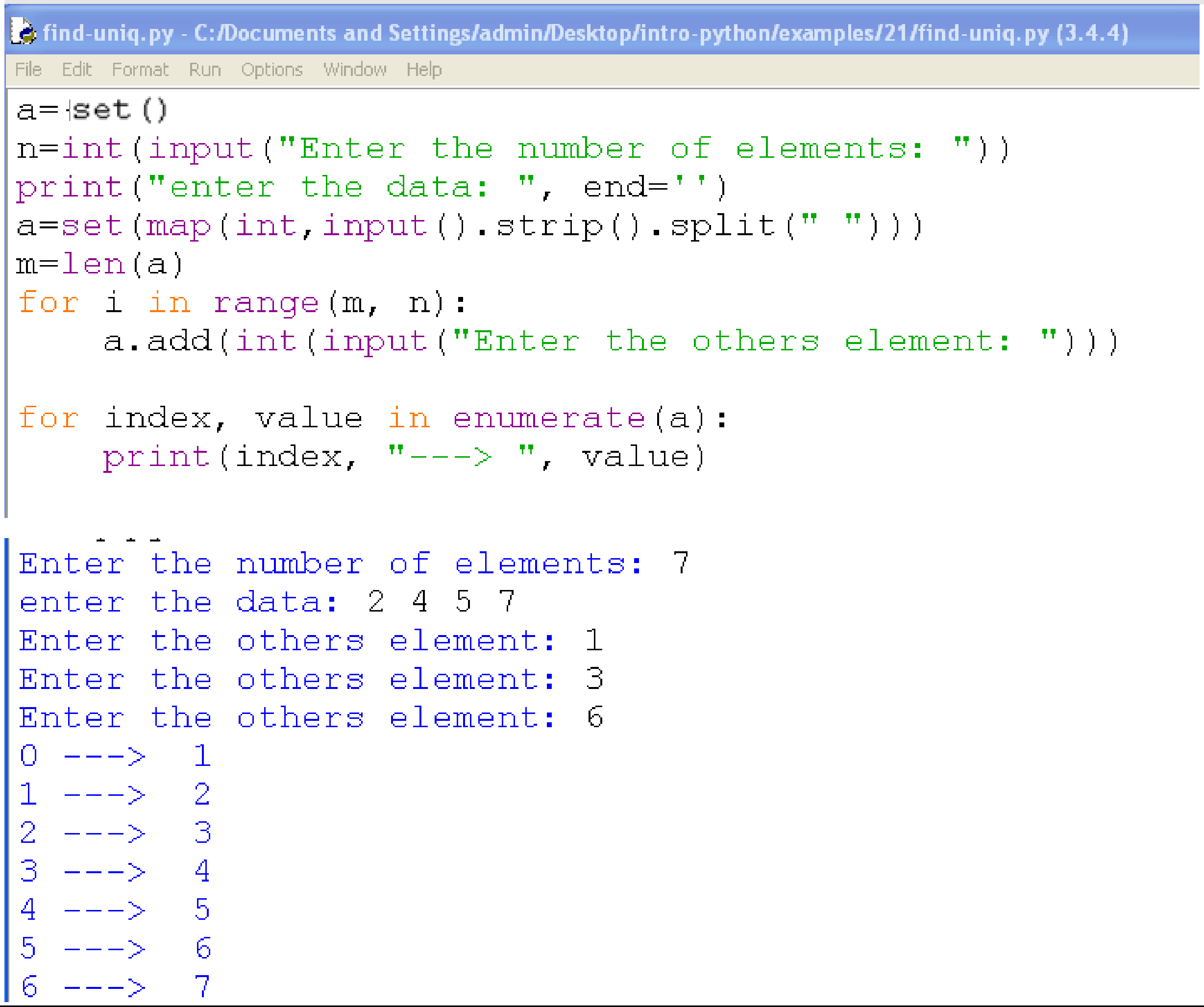
```
cities = {"shiraz","Tehran", ("Mashhad", 12), 25}
for city in cities:
    print(city)


# output

    Tehran
    25
    ('Mashhad', 12)
    shiraz
```

Enumerate and for


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
cities = {"shiraz","Tehran", ("Mashhad", 12), 25}
for pair in enumerate(cities):
    print(pair)

for index, value in enumerate(cities):
    print(index, '...', value)


#  output

    (0, 25)
    (1, ('Mashhad', 12))
    (2, 'Tehran')
    (3, 'shiraz')
    0 ... 25
    1 ... ('Mashhad', 12)
    2 ... Tehran
    3 ... shiraz
```

Enumerate and for


find-uniq.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/find-uniq.py (3.4.4)
File Edit Format Run Options Window Help

```
a=set()
n=int(input("Enter the number of elements: "))
print("enter the data: ", end='')
a=set(map(int,input().strip().split(" ")))
m=len(a)
for i in range(m, n):
    a.add(int(input("Enter the others element: ")))

for index, value in enumerate(a):
    print(index, "---> ", value)
```

```
Enter the number of elements: 7
enter the data: 2 4 5 7
Enter the others element: 1
Enter the others element: 3
Enter the others element: 6
0 --->  1
1 --->  2
2 --->  3
3 --->  4
4 --->  5
5 --->  6
6 --->  7
```

Shared References

```python
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = dict1
print (id(dict1), id(dict2))    # same object
dict2['Name']='ali'
print ("Dictinary one: ", dict1)
print ("Dictinary two: ", dict2)
```

```
140457388081800 140457388081800
Dictinary one:  {'Age': 7, 'Name': 'ali'}
Dictinary two:  {'Age': 7, 'Name': 'ali'}
```

```python
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = {'Name': 'Zara', 'Age': 7};
print (id(dict1), id(dict2))     # different object
dict2['Name']='ali'
print ("Dictinary one: ", dict1)
print ("Dictinary two: ", dict2)
```

```
140457412732232 140457396810632
Dictinary one:  {'Age': 7, 'Name': 'Zara'}
Dictinary two:  {'Age': 7, 'Name': 'ali'}
```

```python
dict1 = {'Name': 'Zara', 'Age': 7}
dict2 = {'Name': 'Zara', 'Age': 7}
print(id(dict1['Name']), id(dict2['Name']))  # same object
```

```
140457388089952 140457388089952
```

Shared References

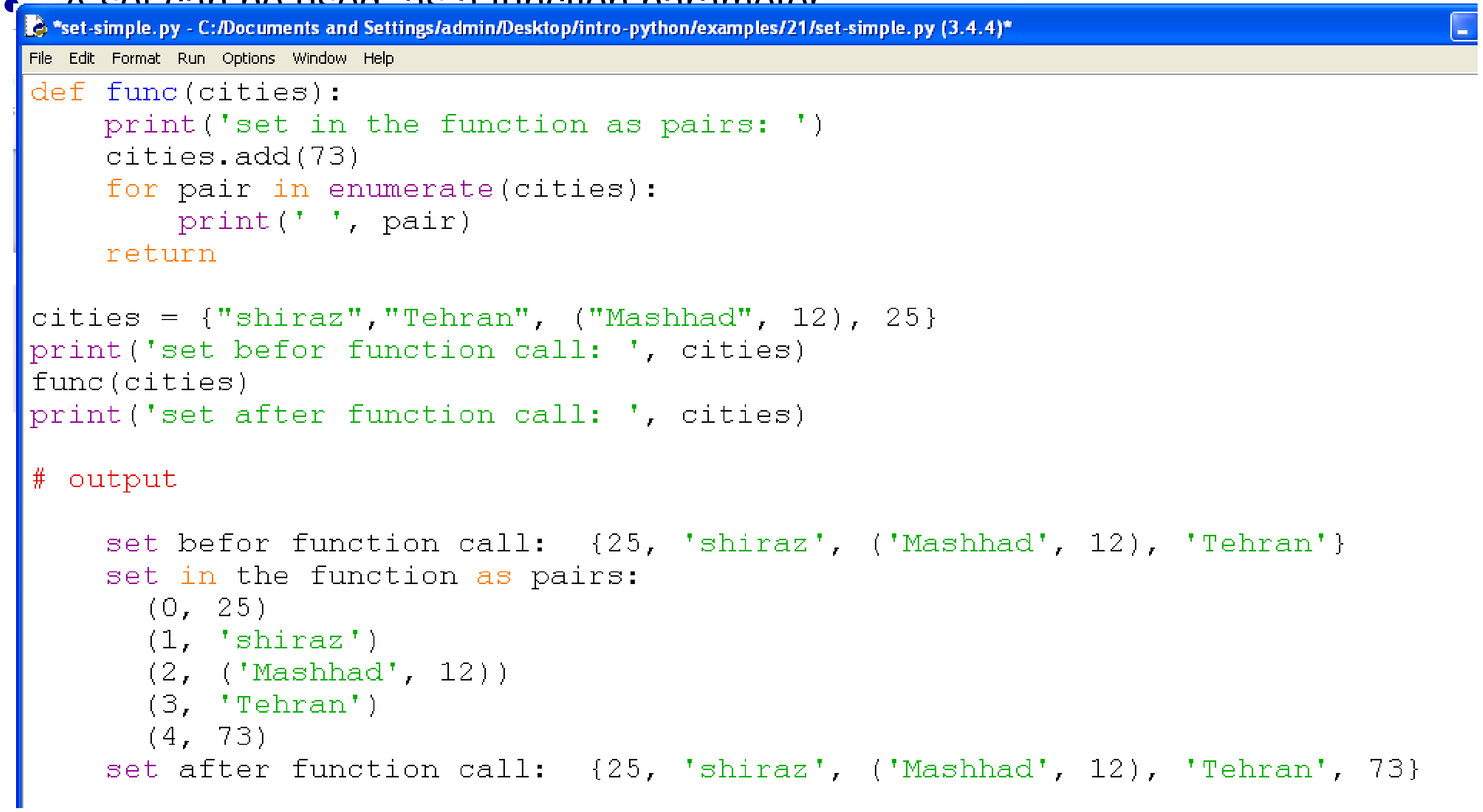
```
s1={1, 3, 5}
s2=s1
print(id(s1), id(s2))

#output

    140497693782760 140497693782760

s1={1, 3, 5}
s2={1, 3, 5}
print(id(s1), id(s2))

#output

    140497693783432 140497693783208
```

Set as a function’s parameter

If you pass set to a function, the passing acts like call-by-reference. Like lists, we have to consider two cases: Elements of a list can be changed in place, i.e. the set will be changed even in the caller's scope. If a new set is assigned to the name, the old list will not be affected, i.e. the set in the caller's scope will remain untouched.

Set as a function's parameter

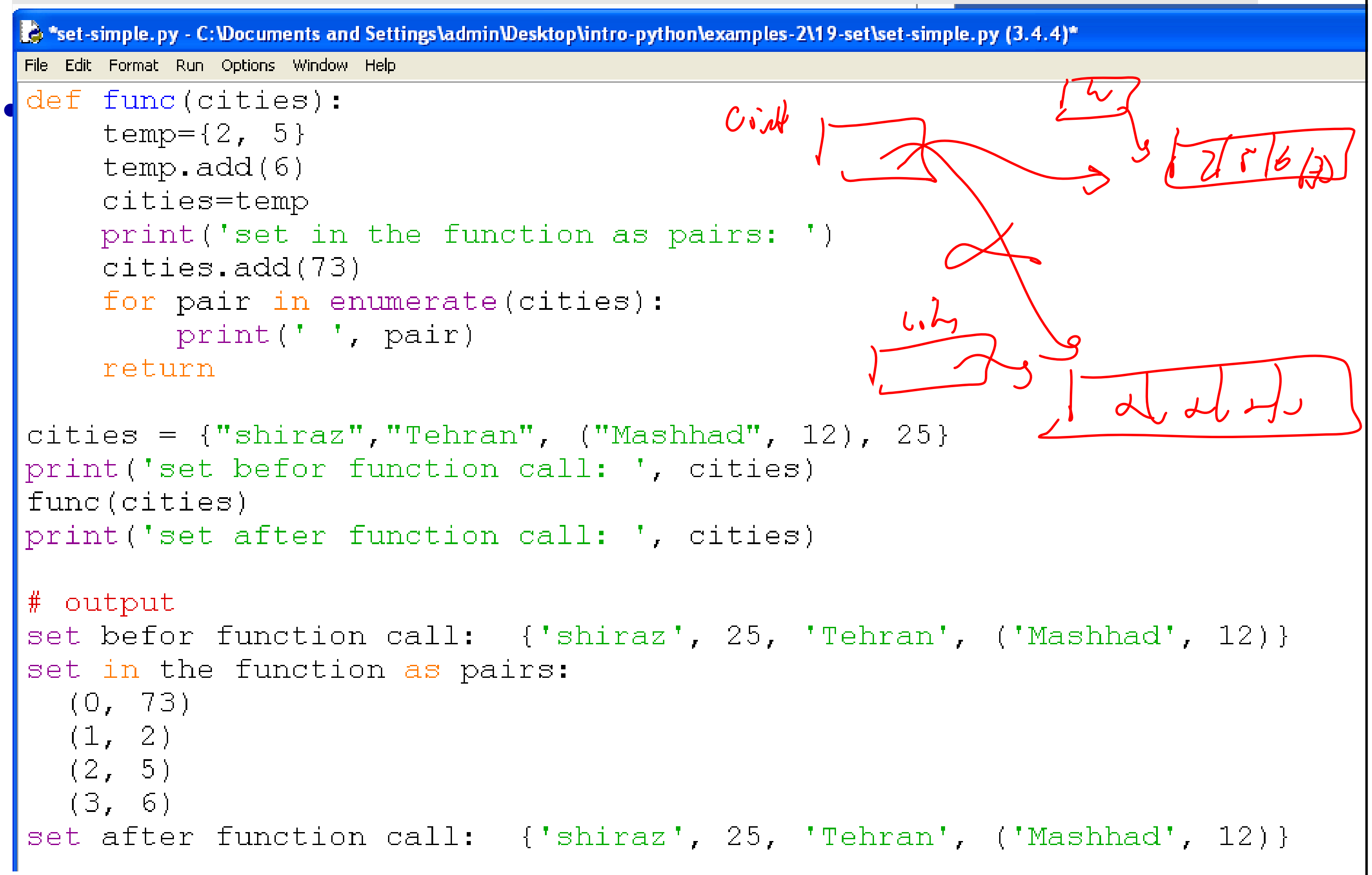
- A set can be used as a function parameter



set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

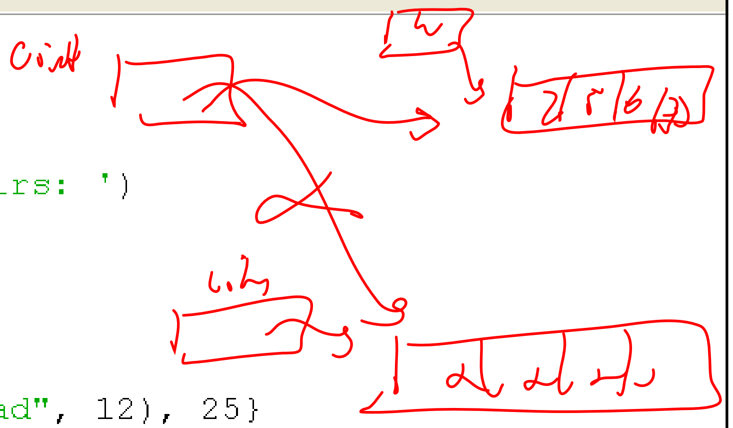
```
def func(cities):
    print('set in the function as pairs: ')
    cities.add(73)
    for pair in enumerate(cities):
        print('  ', pair)
    return

cities = {"shiraz","Tehran", ("Mashhad", 12), 25}
print('set befor function call: ', cities)
func(cities)
print('set after function call: ', cities)

# output

    set befor function call:  {25, 'shiraz', ('Mashhad', 12), 'Tehran'}
    set in the function as pairs:
      (0, 25)
      (1, 'shiraz')
      (2, ('Mashhad', 12))
      (3, 'Tehran')
      (4, 73)
    set after function call:  {25, 'shiraz', ('Mashhad', 12), 'Tehran', 73}
```

Set as a function’s parameter



set-simple.py - C:\Documents and Settings\admin\Desktop\intro-python\examples-2\19-set\set-simple.py (3.4.4)

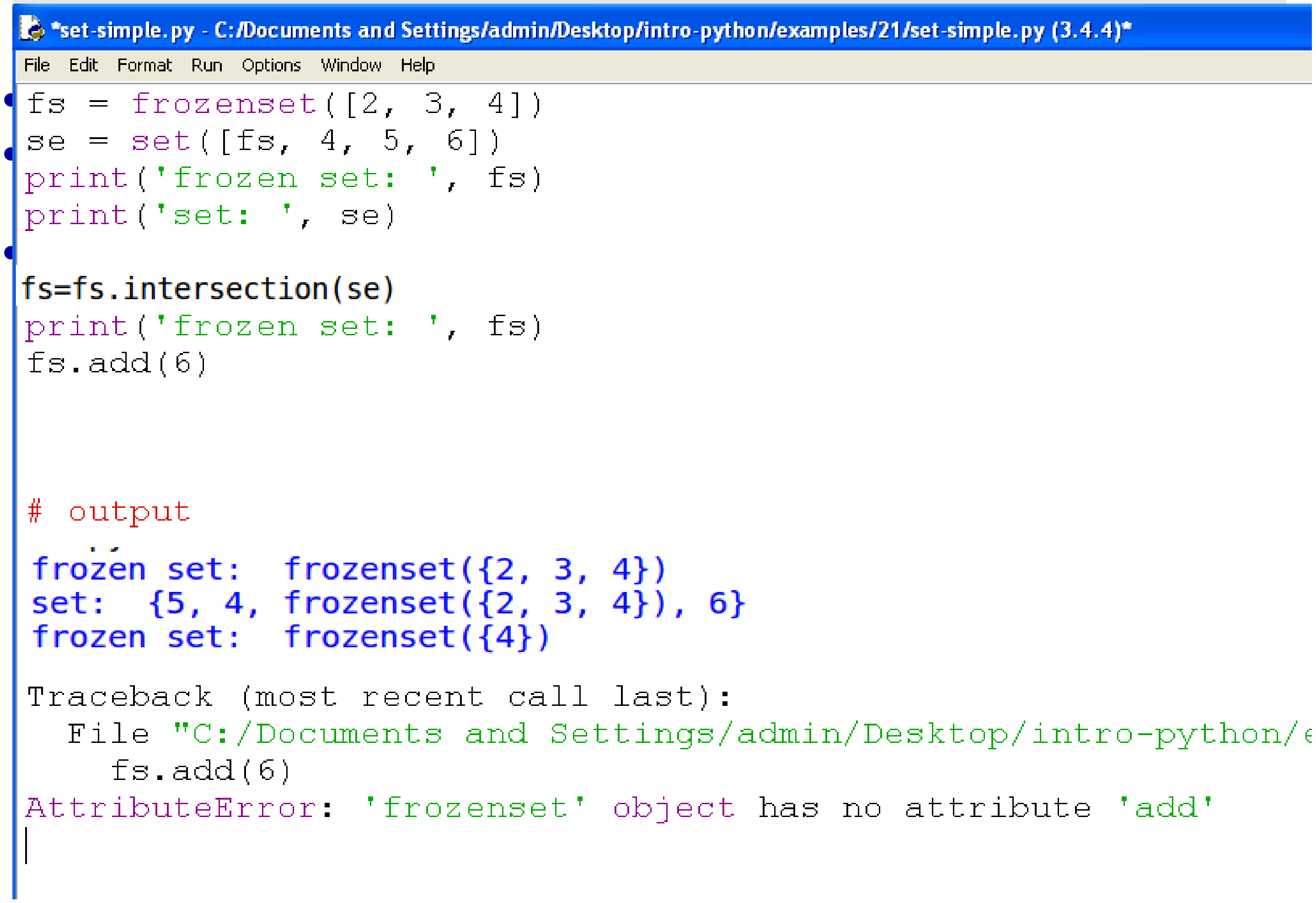
```
def func(cities):
    temp={2, 5}
    temp.add(6)
    cities=temp
    print('set in the function as pairs: ')
    cities.add(73)
    for pair in enumerate(cities):
        print(' ', pair)
    return

cities = {"shiraz","Tehran", ("Mashhad", 12), 25}
print('set befor function call: ', cities)
func(cities)
print('set after function call: ', cities)

# output
set befor function call:  {'shiraz', 25, 'Tehran', ('Mashhad', 12)}
set in the function as pairs:
  (0, 73)
  (1, 2)
  (2, 5)
  (3, 6)
set after function call:  {'shiraz', 25, 'Tehran', ('Mashhad', 12)}
```



frozenset

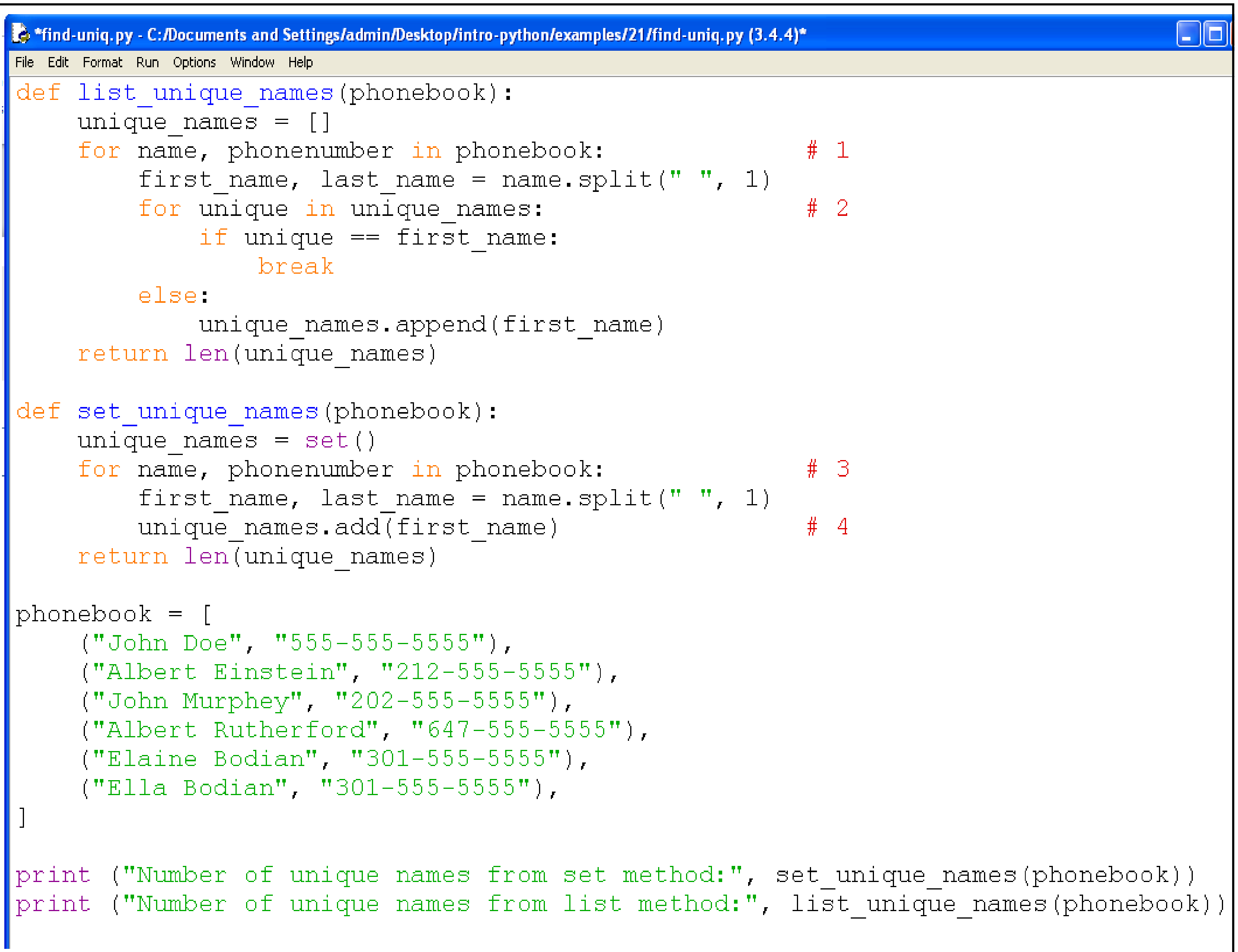
- Though sets can't contain mutable objects, sets are mutable.
- A frozenset is basically the same as a set, except that it is immutable - once it is created, its members cannot be changed.
- Since they are immutable, they are also hashable, which means that frozensets can be used as members in other sets and as dictionary keys. frozensets have the same functions as normal sets, except none of the functions that change the contents (update, remove, pop, etc.) are available.

frozenset


set-simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/set-simple.py (3.4.4)
File Edit Format Run Options Window Help

```
fs = frozenset([2, 3, 4])
se = set([fs, 4, 5, 6])
print('frozen set: ', fs)
print('set: ', se)

fs=fs.intersection(se)
print('frozen set: ', fs)
fs.add(6)



# output

frozen set:  frozenset({2, 3, 4})
set:  {5, 4, frozenset({2, 3, 4}), 6}
frozen set:  frozenset({4})

Traceback (most recent call last):
  File "C:/Documents and Settings/admin/Desktop/intro-python/e
    fs.add(6)
AttributeError: 'frozenset' object has no attribute 'add'
```

Example


find-uniq.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/21/find-uniq.py (3.4.4)
File Edit Format Run Options Window Help

```python
def list_unique_names(phonebook):
    unique_names = []
    for name, phonenumber in phonebook:                 # 1
        first_name, last_name = name.split(" ", 1)
        for unique in unique_names:                     # 2
            if unique == first_name:
                break
        else:
            unique_names.append(first_name)
    return len(unique_names)

def set_unique_names(phonebook):
    unique_names = set()
    for name, phonenumber in phonebook:                 # 3
        first_name, last_name = name.split(" ", 1)
        unique_names.add(first_name)                    # 4
    return len(unique_names)

phonebook = [
    ("John Doe", "555-555-5555"),
    ("Albert Einstein", "212-555-5555"),
    ("John Murphey", "202-555-5555"),
    ("Albert Rutherford", "647-555-5555"),
    ("Elaine Bodian", "301-555-5555"),
    ("Ella Bodian", "301-555-5555"),
]

print ("Number of unique names from set method:", set_unique_names(phonebook))
print ("Number of unique names from list method:", list_unique_names(phonebook))
```

Example

```
Number of unique names from set method: 4
Number of unique names from list method: 4
```

Power set

```
def pset(mset,n):  ⟶ 3
    i=1
    while(i<=n):
       if i not in mset:
           mset=mset | {i}
           print(mset)
           i=1
       else:
           mset=mset - {i}
           i=i+1

n=int(input("Enter the number of elements: "))
a=set()
pset(a,n)
```

```
Enter the number of elements: 3
{1}
→ {2}
→ {1, 2}
→ {3}
→ {1, 3}
{2, 3}
{1, 2, 3}
```

i	mset
1	{1}
1	{}
2	{2}
1	{1, 2}
1	{2}
2	{}
3	{3}
1	{1, 3}
1	

Power set

```python
def pset(mset,n):
    i=n
    while(i!=0):
       if i not in mset:
           mset=mset | {i}
           print(mset)
           i=n
       else:
           mset=mset - {i}
           i=i-1

n=int(input("Enter the number of elements: "))
a=set()
pset(a,n)
```

```
Enter the number of elements: 3
{3}
{2}
{2, 3}
{1}
{1, 3}
{1, 2}
{1, 2, 3}
```