Pair Exercise: Sets

Peter Gyorda April 12, 2025

For DSE5002

Sources https://docs.python.org/2/library/sets.html

Sets are unordered collections of unique objects

They are mutable

sets are defined within curly brackets {}

In [5]:
```
a={1,3,5,7,9,1,1}
a
```

Out[5]:
```
{1, 3, 5, 7, 9}
```

I entered 1 several times, but it only appears once in the set

Values are in the set or not, there is no need to list them more than once

In [9]:
```
#we can iterate a set, but the order may be random

for val in a:
    print(val)
```

```
1
3
5
7
9
```

In [11]:
```
dir(a)
```

Out[11]:
```
['__and__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__iand__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__isub__',
 '__iter__',
 '__ixor__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__rand__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__ror__',
 '__rsub__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__xor__',
 'add',
 'clear',
```

```
 'copy',
 'difference',
 'difference_update',
 'discard',
 'intersection',
 'intersection_update',
 'isdisjoint',
 'issubset',
 'issuperset',
 'pop',
 'remove',
 'symmetric_difference',
 'symmetric_difference_update',
 'union',
 'update']
```

In [13]:
```
#adding to a set
a.add(11)
a
```

Out[13]:
```
{1, 3, 5, 7, 9, 11}
```

In [15]:
```
# there is a pop function, it removes an arbitrary element
z=a.pop()
print(a)
print(z)
```

```
{3, 5, 7, 9, 11}
1
```

In [17]:
```
# add z back!
a.add(z)
```

In [19]:
```
#merging sets
b={2,4,6,8}
a.update(b)
a
```

Out[19]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 11}
```

In [21]:
```
# adding an alias

c=b
c.pop()
print(b)
print(c)
```

```
{2, 4, 6}
{2, 4, 6}
```

In [23]:
```
#making a copy
d=a.copy()
d.pop()
print(d)
print(a)
```

```
{2, 3, 4, 5, 6, 7, 8, 9, 11}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 11}
```

In [25]:
```
#find the length of a set

len(a)
```

Out[25]: 10

Set Operations

In [27]:
```
# testing for membership

y=21
y in a
```

Out[27]: False

In [29]:
```
y not in a
```

Out[29]: True

In [31]:
```
#classic set operations

t={"apple","orange","banana"}
```

```
c={"red","green","orange"}

#union
t|c
```

Out[31]: {'apple', 'banana', 'green', 'orange', 'red'}

In [33]:
```
#intersection
t&c
```

Out[33]: {'orange'}

In [35]:
```
#set differences
t-c
```

Out[35]: {'apple', 'banana'}

In [37]:
```
c-t
```

Out[37]: {'green', 'red'}

Why is c-t not equal to c-t? What is going on here?

look up symmetric _difference for python sets

Add a markdown cell and explain this

In [ ]:
```
c - t (Set Difference):
c - t finds elements in c that are not in t.
In this case, c - t results in {'red', 'green'} because "red" and "green" are in c but not in t, while "orange" is i
The operation is deterministic: c - t will always produce {'red', 'green'} given the same c and t. Therefore c - t =
c ^ t (Symmetric Difference):
c ^ t finds elements that are in either c or t, but not in their intersection.
The result is {'apple', 'banana', 'red', 'green'}.
"orange" is excluded because it's in both sets.
The symmetric difference can also be written as c.symmetric_difference(t)
```

What are sets good for?

Testing for membership

Sets work much faster for the "in" test, since sets are hashed.

the "in" test can be done using a list, but this is not hashed and thus 50 to 100 times slowere

Not a big deal for a small project, a huge deal for a bit data set that is repeatedly re-analyzed