Lecture 4.3 : Sets

Introduction

- A *set* is a *collection* of objects of arbitrary type.
- The objects in a set are called its *members*.
- A key property of a set is that it may contain only one copy of a particular object: duplicates are not allowed.
- A set with no elements is *the empty set*.

Python sets

- A set is created by calling the `set()` constructor or by using curly brackets.
- Note a `dictionary` is also created using curly brackets but while a dictionary consists of *key-value* pairs separated by a colon, the members of a `set` are separated by commas.
- The `set()` constructor requires an iterable be passed to it. Each object in the iterable becomes a member of the set.
- There is no order to the members of a set (like a dictionary).

```
vowel_set = set('aeiou') # vowel_set = {'aeiou'} is equivalent
print(vowel_set)
print(len(vowel_set))
```

```
{'e', 'u', 'o', 'a', 'i'}
5
```

- A set cannot contain duplicates (duplicates in the original iterable are absent in the corresponding set).

```
number_set = set([1, 2, 3, 1, 2, 7])
print(number_set)
```

```
{1, 2, 3, 7}
```

```
s = "Some characters"
char_set = set(s)
print(char_set)
print(len(char_set))
print(len(s))
```

```
{'e', 'h', 'm', 'o', 's', 'r', ' ', 'c', 'a', 'S', 't'}
11
15
```

Set membership

- We can use the `in` operator to check for membership of a `set`.

```
print(vowel_set)
print('a' in vowel_set)
print('x' in vowel_set)
```

```
{'e', 'u', 'o', 'a', 'i'}
True
False
```

Iteration over a set

- We can use a `for` loop to iterate over the members of a `set`.
- Since a `set` is unordered, the order in which members are visited by a `for` loop is unknown (and you should not write code that relies on the order).

```
print(vowel_set)
for v in vowel_set:
   print(v)
```

```
{'e', 'u', 'o', 'a', 'i'}
e
u
o
a
i
```

Set methods

- We can remove an item from a set with `remove()` and add an item with `add()`.

```
vowel_set = set('aeiou')
vowel_set.remove('u')
print(vowel_set)
vowel_set.add('u')
print(vowel_set)
```

```
{'e', 'o', 'a', 'i'}
{'e', 'u', 'o', 'a', 'i'}
```

- We can find the *intersection* of two sets using the `intersection()` method.

- The intersection of sets A and B is the set of elements that are in both A and B.

```
a_set = set('adcb')
b_set = set('ecdf')
print(a_set.intersection(b_set))
print(a_set & b_set) # equivalent to above
```

```
{'d', 'c'}
{'d', 'c'}
```

- We can find the *union* of two sets using the `union()` method.
- The union of sets A and B is the set of elements that are in A or B.

```
a_set = set('adcb')
b_set = set('ecdf')
print(a_set.union(b_set))
print(a_set | b_set) # equivalent to above
```

```
{'e', 'f', 'b', 'd', 'c', 'a'}
{'e', 'f', 'b', 'd', 'c', 'a'}
```

- We can find the *set difference* between two sets using the `difference()` method.
- A set difference B is the set of elements in A but not in B.
- B set difference A is the set of elements in B but not in A.

```
a_set = set('adcb')
b_set = set('ecdf')
print(a_set.difference(b_set))
print(a_set - b_set) # equivalent to above
print(b_set.difference(a_set))
print(b_set - a_set) # equivalent to above
```

```
{'a', 'b'}
{'a', 'b'}
{'e', 'f'}
{'e', 'f'}
```

- We can check whether one set is a *subset* of another using the `issubset()` method.
- Set A is a subset of set B if every member of A is also a member of B.

```
a_set = set('abcd')
b_set = set('bd')
print(a_set.issubset(b_set))
print(a_set <= b_set) # equivalent to above
print(b_set.issubset(a_set))
print(b_set <= b_set) # equivalent to above
```

```
False
False
True
True
```

- We can check whether one set is a *superset* of another using the `issuperset()` method.
- Set A is a superset of set B if every member of B is also a member of A.

```
a_set = set('abcd')
b_set = set('bd')
print(a_set.issuperset(b_set))
print(a_set >= b_set) # equivalent to above
print(b_set.issuperset(a_set))
print(b_set >= a_set) # equivalent to above
```

```
True
True
False
False
```

Examples

- Write a function that returns `True` if a string contains a vowel and `False` otherwise.

```
vowel_set = set('aeiou')
def contains_vowel(s):
    return len(vowel_set & set(s.lower())) > 0

print(contains_vowel('Owl'))
print(contains_vowel('lynx'))
```

```
True
False
```

- Write a function that returns the number of unique vowels in a sentence.

```
vowel_set = set('aeiou')
def unique_vowels(s):
    return len(vowel_set & set(s.lower()))

print(unique_vowels('Owl'))
print(unique_vowels('lynx'))
print(unique_vowels('Oodles of poodles'))
```

```
1
0
```

```
2
```

- Write a function that returns `True` if a string contains all the vowels and `False` otherwise.

```
vowel_set = set('aeiou')
def contains_all_vowels(s):
   return vowel_set & set(s.lower()) == vowel_set

print(contains_all_vowels('sequioa'))
print(contains_all_vowels('oak'))
```

```
True
False
```

- Write a function that returns `True` if a list of numbers contains duplicates and `False` otherwise.

```
def has_duplicates(numbers):
   return len(numbers) > len(set(numbers))

print(has_duplicates([3,1,9,7,4]))
print(has_duplicates([3,1,4,7,4]))
```

```
False
True
```

- Write a function that takes a single string `s` as its argument and returns a dictionary mapping each character in `s` to the number of times it occurs in `s`. Ignore case.

```
def counter(s):
   sl = s.lower()

   # make use of a dictionary comprehension
   d = {c : sl.count(c) for c in set(sl)}
   return d

print(counter('Totally'))
print(counter(''))
```

```
{'o': 1, 'y': 1, 'l': 2, 'a': 1, 't': 2}
{}
```

Set methods

```
help(set)
```

```
Help on class set in module builtins:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __isub__(self, value, /)
 |      Return self-=value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __ixor__(self, value, /)
 |      Return self^=value.
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __or__(self, value, /)
 |      Return self|value.
 |
 |  __rand__(self, value, /)
 |      Return value&self.
 |
 |  __reduce__(...)
```

```
|      Return state information for pickling.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(...)
|      S.__sizeof__() -> size of S in memory, in bytes
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __xor__(self, value, /)
|      Return self^value.
|
|  add(...)
|      Add an element to a set.
|
|      This has no effect if the element is already present.
|
|  clear(...)
|      Remove all elements from this set.
|
|  copy(...)
|      Return a shallow copy of a set.
|
|  difference(...)
|      Return the difference of two or more sets as a new set.
|
|      (i.e. all elements that are in this set but not the others.)
|
|  difference_update(...)
|      Remove all elements of another set from this set.
|
|  discard(...)
|      Remove an element from a set if it is a member.
|
|      If the element is not a member, do nothing.
|
|  intersection(...)
|      Return the intersection of two sets as a new set.
|
|      (i.e. all elements that are in both sets.)
|
|  intersection_update(...)
|      Update a set with the intersection of itself and another.
|
|  isdisjoint(...)
|      Return True if two sets have a null intersection.
|
|  issubset(...)
|      Report whether another set contains this set.
|
|  issuperset(...)
|      Report whether this set contains another set.
|
|  pop(...)
|      Remove and return an arbitrary set element.
|      Raises KeyError if the set is empty.
```

```
 |
 |  remove(...)
 |      Remove an element from a set; it must be a member.
 |
 |      If the element is not a member, raise a KeyError.
 |
 |  symmetric_difference(...)
 |      Return the symmetric difference of two sets as a new set.
 |
 |      (i.e. all elements that are in exactly one of the sets.)
 |
 |  symmetric_difference_update(...)
 |      Update a set with the symmetric difference of itself and another.
 |
 |  union(...)
 |      Return the union of sets as a new set.
 |
 |      (i.e. all elements that are in either set.)
 |
 |  update(...)
 |      Update a set with the union of itself and others.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```