Dictionaries and Sets

Dictionaries and Sets

- Dictionaries map keys to values
- Sets represent mathematical sets

Byt the end of this lesson you will

- know how to use dictionaries
- know jow to use sets

Dictionaries

A dictionary is a map from keys to values.

Create dictionaries with {}

```
>>> capitals = {}
```

Add key-value pairs with assignment operator

```
>>> capitals['Georgia'] = 'Atlanta'
>>> capitals['Alabama'] = 'Montgomery'
>>> capitals
{'Georgia': 'Altanta', 'Alabama': 'Montgomery'}
```

Keys are unique, so assignment to same key updates mapping

```
>>> capitals['Alabama'] = 'Birmingham'
>>> capitals
{'Georgia': 'Altanta', 'Alabama': 'Birmingham'}
```

Dictionary Operations

Remove a key-value mapping with `del` statement

```
>>> del capitals['Alabama']
>>> capitals
{'Georgia': 'Atlanta'}
```

Use the `in` operator to test for existence of key (not value)

```
>>> 'Georgia' in capitals
True
>>> 'Atlanta' in capitals
False
```

Extend a dictionary with `update()` method, get values as a list with values method

```
>>> capitals.update({'Tennessee': 'Nashville', 'Mississippi':
'Jackson'})
>>> capitals.values()
dict_values(['Jackson', 'Nashville', 'Atlanta'])
```

Conversions to `dict`

Any sequence of two-element sequences can be converted to a `dict`

A list of two-element lists:

```
>>> dict([[1, 1], [2, 4], [3, 9], [4, 16]])
{1: 1, 2: 4, 3: 9, 4: 16}
```

A list of two-element tuples:

```
>>> dict([('Lassie', 'Collie'), ('Rin Tin Tin', 'German
Shepherd')])
{'Rin Tin Tin': 'German Shepherd', 'Lassie': 'Collie'}
```

Even a list of two-character strings:

```
>>> dict(['a1', 'a2', 'b3', 'b4'])
{'b': '4', 'a': '2'}
```

Notice that subsequent pairs overwrote previously set keys.

Sets

Sets have no duplicates, like the keys of a `dict`. They can be iterated over (we'll learn that later) but can't be accessed by index.

- Create an empty set with `set()` function, add elements with `add()` method

```
>>> names = set()
>>> names.add('Ally')
>>> names.add('Sally')
>>> names.add('Mally')
>>> names.add('Ally')
>>> names
{'Ally', 'Mally', 'Sally'}
```

- Converting to set a convenient way to remove duplicates

```
>>> set([1,2,3,4,3,2,1])
{1, 2, 3, 4}
```

Set Operations

Intersection (elements in a and b)

```
>>> a = {1, 2}
>>> b = {2, 3}
>>> a & b # or a.intersetion(b)
{2}
```

Union (elements in a or b)

```
>>> a | b # or a.union(b)
{1, 2, 3}
```

Set Operations

Difference (elements in a that are not in b)

```
>>> a - b # or a.difference(b)
{1}
```

Symmetric difference (elements in a or b but not both)

```
>>> a ^ b # or a.symmetric_difference(b)
{1, 3}
```

Set Predicates

A predicate function asks a question with a `True` or `False` answer.

Subset of:

```
>>>a <= b # or a.issubset(b)
False
```

Proper subset of:

```
>>> a < b
False
```

Set Predicates

Superset of:

```
>>> a >= b # or a.issuperset(b)
False
```

Proper superset of:

```
>>> a > b
False
```

Closing Thoughts

Typical Python programs make extensive use of built-in data structures and often combine them (lists of lists, dictionaries of lists, etc)

- These are just the basics
- Explore these data structures on your own
- Read the books and Python documentation

This is a small taste of the expressive power and syntactic convenience of Python's data structures.