Lecture 4.1 : Dictionaries 1¶

Introduction

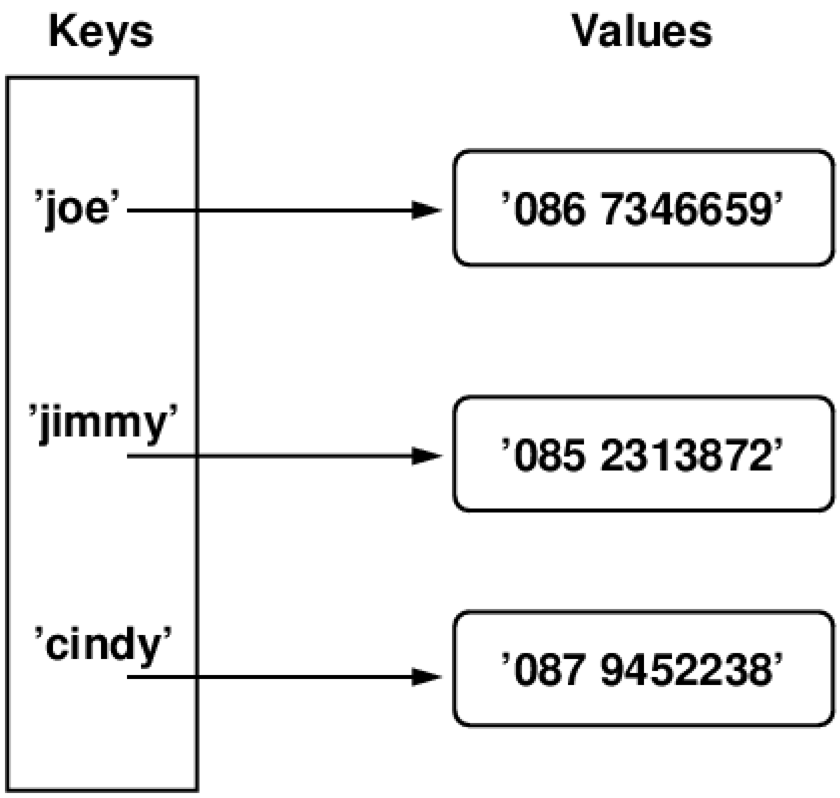
- So far we have met lists, strings and tuples. Each of these is an example of a *data structure*.
- Here we examine another built-in Python data structure: the *dictionary*.
- As we will see dictionaries are an extremely useful and powerful data structure. Knowing when and how to use them effectively will make you a better programmer.

Dictionaries

- A dictionary is a *collection type* but it is **not** a *sequence type*. That is, its elements are not ordered as they are in a list, string or tuple.
- What is called a dictionary in Python is also sometimes referred to as a *map*, *hashmap*, or *associative array* in other programming languages.
- We can think of a dictionary as a collection of pairs of objects. One element in the pair is the *key* and the other is the *value*. A dictionary thus implements a *mapping* from keys to values.
- When we use a real world dictionary to look up the meaning of a word, the *word* is the *key* and the *meaning of the word* is the *value*.
- A dictionary is designed such that given a *key*, retrieving the associated *value* is a highly efficient operation.
- Once a dictionary has been created we can make changes to the values it contains, add new key-value mappings and remove existing key-value mappings. Clearly a dictionary is a *mutable* type.
- We do not typically know the order in which key-value pairs are stored in a Python dictionary. That information is hidden from us and we should not write programs that rely on it.

Dictionary example

- We can use a dictionary to implement a simple phone book. A phone book is a mapping from names to phone numbers.
- The dictionary keys are thus names and the dictionary values are phone numbers.
- Once built, the dictionary can be depicted as shown below.



- To find Cindy's phone number in the dictionary we use the key `'cindy'`. That leads us to the value `'087 9452238'`.

Building dictionaries

- While we use square brackets to create a list, we use curly brackets to create a dictionary.
- Key-value pairs are separated by a colon.
- Keys must be of an immutable type (e.g. strings, integers, tuples) but values can be of any type.
- Let's create the dictionary depicted above.

```
phone_book = { 'joe' : '086 7346659',
   'jimmy' : '085 2313872',
   'cindy' : '087 9452238' }

print(phone_book)
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238'}
```

Dictionary indexing

- Dictionaries are indexed by keys.

```
print(phone_book['cindy'])
print(phone_book['jimmy'])
```

```
087 9452238
085 2313872
```

- It is an error to index a dictionary with a non-existent key. Specifically, a `KeyError` exception is thrown.

```
print(phone_book['sally'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [3], in <module>
----> 1 print(phone_book['sally'])

KeyError: 'sally'
```

- Note dictionaries are *not* sequenced and cannot be indexed by position (as immutable types integers can serve as keys but even then we are indexing by key and not by position).

```
print(phone_book[0])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [4], in <module>
----> 1 print(phone_book[0])

KeyError: 0
```

Dictionary assignment

- To add an additional mapping to an existing dictionary we use square brackets to index by the new key and then assign the new value (note how *where* a new entry goes in the dictionary is not in general predictable).

```
print(phone_book)
phone_book['louie'] = '087 6551201'
print(phone_book)
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238'}
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238', 'louie': '087 6
```

Dictionary updates

- To update an existing key-value pair in the dictionary we index by key and supply the new value.

```
print(phone_book)
phone_book['louie'] = '086 6551201'
print(phone_book)
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238', 'louie': '087 6
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238', 'louie': '086 6
```

Dictionary deletions

- To remove an existing key-value pair in the dictionary we use `del()`.

```
print(phone_book)
# so long louie
del(phone_book['louie'])
print(phone_book)
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238', 'louie': '086 6
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238'}
```

Avoiding KeyErrors

- We have a number of options available in order to avoid KeyErrors.

```
def lookup(name):
    # Check membership with in
    if name in phone_book:
        return phone_book[name]
    return None

print(lookup('jimmy'))
print(lookup('sally'))
```

```
085 2313872
None
```

```
def lookup(name):
    try:
        return phone_book[name]
    except KeyError:
        return None

print(lookup('jimmy'))
print(lookup('sally'))
```

```
085 2313872
None
```

```python
def lookup(name):
   # get returns None if key not present
   return phone_book.get(name)

print(lookup('jimmy'))
print(lookup('sally'))
```

```
085 2313872
None
```

Fancy value types

- While keys must be an immutable type values can be any type (strings, tuples, lists, even dictionaries).

```python
address_book = { 'joe' : ('Dublin', 'Ireland'),
   'henri' : ('Paris', 'France') }

print(address_book['joe'])
print(address_book['henri'][0])
print(address_book['henri'][1])
```

```
('Dublin', 'Ireland')
Paris
France
```

Dictionary size

- The `len()` function returns the number of key-value pairs in a dictionary.

```python
print(phone_book)
print(len(phone_book))
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238'}
3
```

Dictionary methods

- We can retrieve a list of all of a dictionary's keys using the `keys()` method.
- We can retrieve a list of all of a dictionary's values using the `values()` method.
- Keys and values returned by the `keys()` and `values()` methods are in corresponding order.

```
print(phone_book)
print(phone_book.keys())
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238'}
dict_keys(['joe', 'jimmy', 'cindy'])
```

```
print(phone_book)
print(phone_book.values())
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238'}
dict_values(['086 7346659', '085 2313872', '087 9452238'])
```

- We can retrieve a list of key-value pairs (tuples) from a dictionary using the `items()` method.

```
print(phone_book)
print(phone_book.items())
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238'}
dict_items([('joe', '086 7346659'), ('jimmy', '085 2313872'), ('cindy', '087 9452238')
```

Iterating over a dictionary

- We can use a `for` loop to iterate over the items in a dictionary (key-value pairs) and use multiple assignment to handily access every key and corresponding value.

```
print(phone_book)
print(phone_book.items())
for k, v in phone_book.items():
    print(f'{k} ---> {v}')
```

```
{'joe': '086 7346659', 'jimmy': '085 2313872', 'cindy': '087 9452238'}
dict_items([('joe', '086 7346659'), ('jimmy', '085 2313872'), ('cindy', '087 9452238')
joe ---> 086 7346659
jimmy ---> 085 2313872
cindy ---> 087 9452238
```