1 Dictionaries

Dictionaries store objects such as text or numbers (or other Python objects). We saw that lists stored elements in sequence, and that each element can be referred to by an index number, which is the position of that element in a list. The contents of dictionaries are accessed by a unique *key* which may be a number or may be text.

Dictionaries take the form of key1:value1, key2:value2

The *value* may be a single number or word, or may be another Python object such as a list or a tuple.

Dictionaries allow fast random-access to data.

1.1 Creating and adding to dictionaries

Dictionaries may be created with content, and are referred to by their *key*:

```
fellowship = {'Man': 2, 'Hobbit':4, 'Wizard':1, 'Elf':1, 'Dwarf':1}
print (fellowship['Man'])

OUT:
2
```

Dictionaries may also be created empty. New contened my be added to an an empty or an existing dictionary.

```
trilogy = {}
trilogy['Part 1'] = 'The fellowshipp of the rring'
trilogy['Part 2'] = 'The Two Towwwwers'
trilogy['Part 3'] = 'The return of the kingg'
print (trilogy['Part 2'])

OUT:
The Two Towwwwers
```

Dictionaries may contain a mixture of data types.

```
lord_of_the_rings = {'books':3,
                     'author':'JRR Tolkien',
                     'main_characters':['Frodo','Sam','Gandalf']}
print (lord_of_the_rings['main_characters'])

OUT:
['Frodo', 'Sam', 'Gandalf']
```

1.2 Changing a dictionary entry

Dictionary elements are over-written if a key already exists.

```
trilogy['Part 2'] = 'The Two Towers'
print (trilogy['Part 2'])

OUT:
The Two Towers
```

Dictionary items may also be updated using another dictionary:

```
entry_update = {'Part 1':'The Fellowship of the Ring',
               'Part 3':'The Return of the King'}
trilogy.update(entry_update)
print (trilogy)
```

```
OUT:
{'Part 1': 'The Fellowship of the Ring', 'Part 2': 'The Two Towers', 'Part 3': 'The Return of the Ki
```

1.3 Deleting a dictionary entry

Dictionary entries may be deleted, by reference to their key, with the *del* command. If the key does not exist an error will be caused, so in the example below we check that the entry exists before trying to delete it.

```
entry_to_delete = 'Part 2'
if entry_to_delete in trilogy:
    del trilogy[entry_to_delete]
    print (trilogy)
else:
    print('That key does not exist')

OUT:
{'Part 1': 'The Fellowship of the Ring', 'Part 3': 'The Return of the King'}
```

1.4 Iterating through a dictionary

There are two main methods to iterate through a dictionary. The first retrieves just the key, and the value may then be retrieved from that key. The second method retrieves the key and value together.

```
lord_of_the_rings = {'books':3,
                     'author':'JRR Tolkien',
                     'main_characters':['Frodo','Sam','Gandalf']}

# Iterating method 1
for key in lord_of_the_rings:
    print (key, '-', lord_of_the_rings[key])

# Iteraring method 2
print()
for key, value in lord_of_the_rings.items():
    print (key,'-', value)

OUT:
books - 3
author - JRR Tolkien
main_characters - ['Frodo', 'Sam', 'Gandalf']

books - 3
author - JRR Tolkien
main_characters - ['Frodo', 'Sam', 'Gandalf']
```

1.5 Converting dictionaries to lists of keys and values

Dictionary keys and values may be accessed separately. We can also return a list of tuples of keys and values with the items method.

```
lord_of_the_rings = {'books':3,
                     'author':'JRR Tolkien',
                     'main_characters':['Frodo','Sam','Gandalf']}
# print the keys:
print ('\nKeys:') # \n creates empty line before text
print (list(lord_of_the_rings.keys()))

# print the values
print ('\nValues:')
print (list(lord_of_the_rings.values()))

# print keys and values
print ('\nKeys and Values:')
print (list(lord_of_the_rings.items()))

OUT:
Keys:
['books', 'author', 'main_characters']

Values:
[3, 'JRR Tolkien', ['Frodo', 'Sam', 'Gandalf']]

Keys and Values:
[('books', 3), ('author', 'JRR Tolkien'), ('main_characters', ['Frodo', 'Sam', 'Gandalf'])]
```

1.6 Using the get method to return a null value if a key does not exist

Using the usual method of referring to a dictionary element, an error will be returned if the key used does not exist. An 'if key in dictionary' statement could be used to check the key exists. Or the get method will return a null value if the key does not exist. The get method allows for a default value to be returned if a key is not found.

```
print (lord_of_the_rings.get('author'))
print (lord_of_the_rings.get('elves'))
print (lord_of_the_rings.get('books','Key not found - sorry'))
print (lord_of_the_rings.get('dwarfs','Key not found - sorry'))

JRR Tolkien
None
3
Key not found - sorry
```

1.7 Intersection between dictionaries

Like sets the intersection between dictionaries may be evaluated. See the entry on sets for more intersection methods.

```
a = {'x':1,'y':2,'z':3,'zz':2}
b = {'x':10,'w':11,'y':2,'yy':2}
# Show keys in common
print('Keys in common: ',a.keys() & b.keys())
# Show keys in a not in b
print('Keys in a but not b:',a.keys()-b.keys()) # this subtract method works with sets {} but not li
# Show keys+values in common (items() return key+value, so both need to be the same)
print('Keys and values in common:',a.items() & b.items())

OUT:
Keys in common:  {'x', 'y'}
Keys in a but not b: {'z', 'zz'}
Keys and values in common: {('y', 2)}
```

1.8 Other dictionary methods

dictionary.clear() empties a dictionary

dictionary.pop(key) will return an item and remove it from the dictionary

1.9 Ordered dictionaries

Like sets, dictionaries do not usually keep the order of entries. If it is useful to keep the order of entry in a dictionary then the OrderDict object is useful. This needs an import statement before it is used (all import statements are usually kept together at the start of code).

```
from collections import OrderedDict
d=OrderedDict()

d['Gandalf']=1
d['Bilbo']=2
d['Frodo']=3
d['Sam']=4

for key in d:
    print(key,d[key])

OUT:
Gandalf 1
Bilbo 2
Frodo 3
Sam 4
```