Dictionaries

Dictionary

- Dictionaries are Python‟s most powerful data collection
- **Definition:**

  A dict is an **unordered collection** of zero or more **key–value pairs** whose **keys** are object references that refer to immutable objects, and whose **values** are object references referring to objects of any type.

  Syntax: dict1 = { key1: Value1, Key2: Value2, …., Keyn: Valuen}

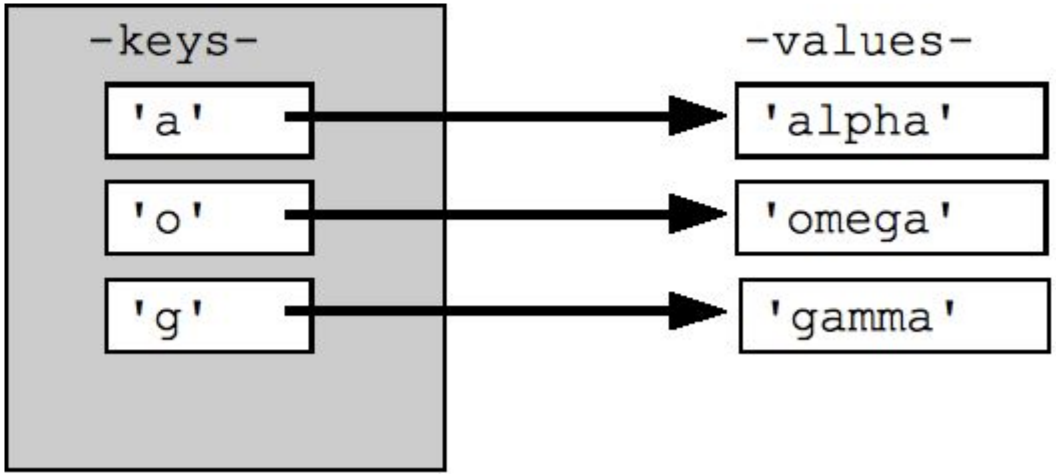


Image Ref: https://cs.stanford.edu/people/nick/python-in-one-easy-lesson/

Dictionary….

Dictionary is like a bag of objects – No order, but we have tag (key) to access

Tags

In my bag there is a **blue crayon**.

In my bag there is a **green book.**

In my bag there is **a notebook.**

In my bag there is **one ruler.**

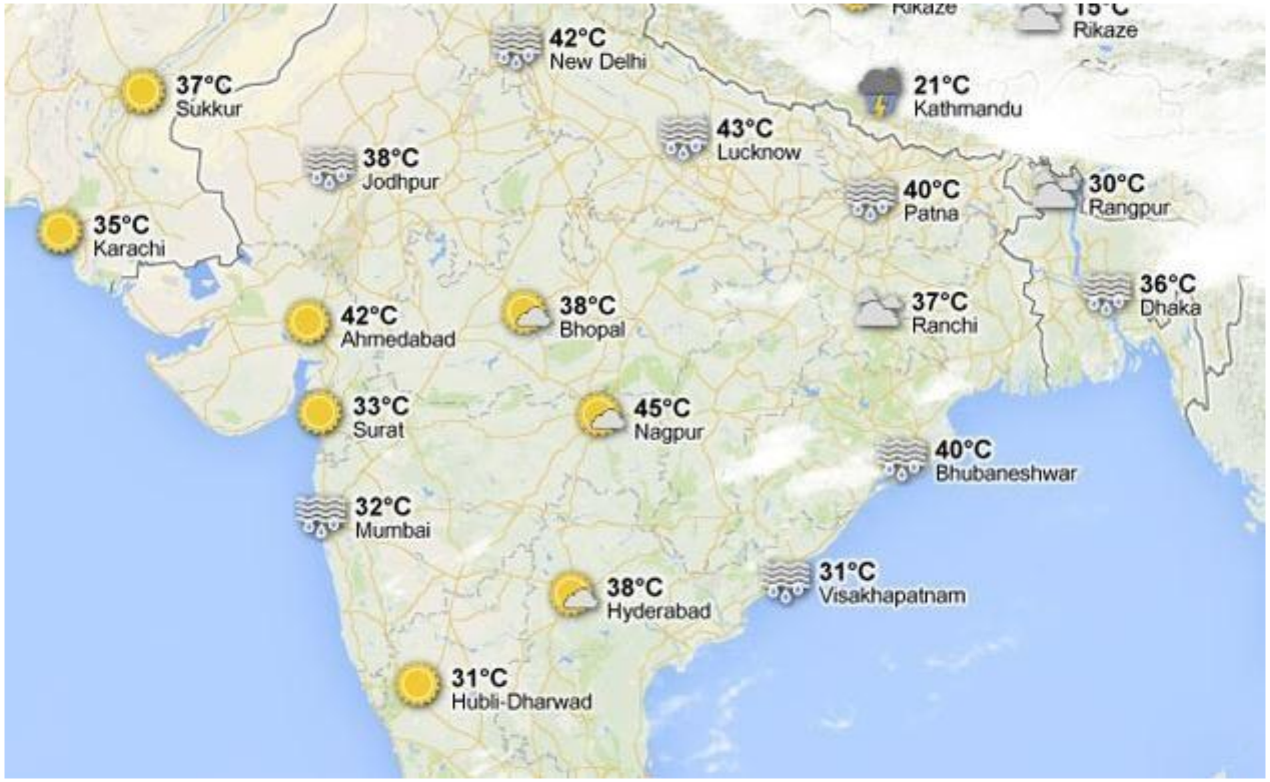
In my bag there is a **pink pencil.**

In my bag there is **an apple.**

Image Credits:
http://www.englishexercises.org/makeagame/viewgame.asp?id=9705

Dictionary - Example

Temperatures of Indian cities as a dictionary



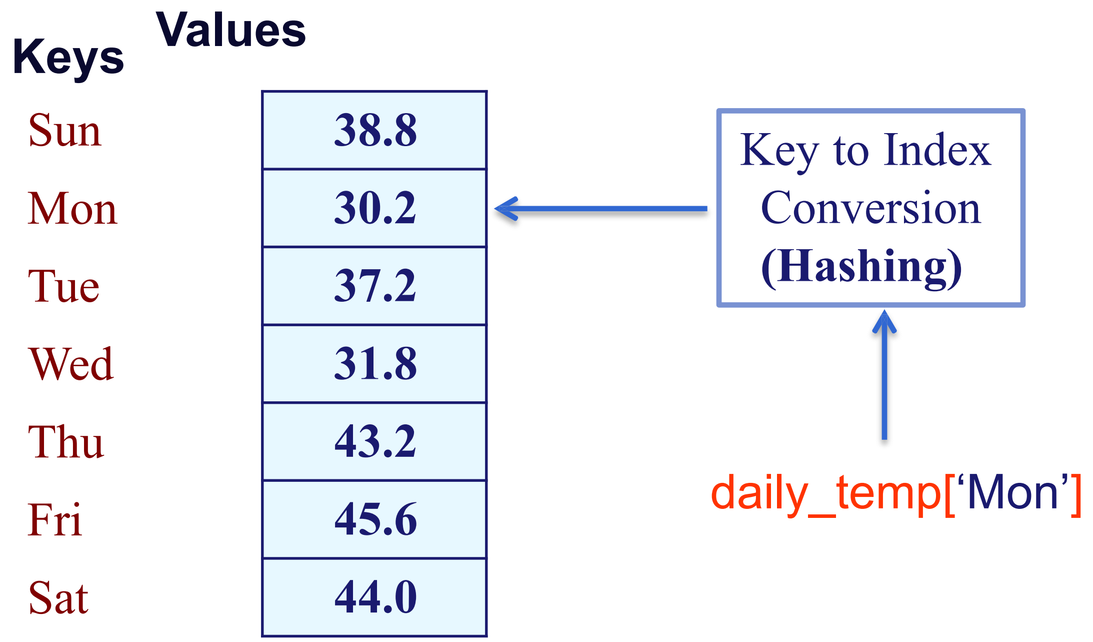
```
>>>city_temp =  {'Mumbai': 32, 'Hyderabad': 38,
                 'Vishakhapatnam': 31, 'Delhi': 42}
```

Image Credits: http://www.ibnlive.com/news/

Dictionary - Example

Example:

```
>>>daily_temp =  {'sun': 38.8, 'mon': 30.2,
                  'tue': 37.2, 'wed': 31.8,
                  'thur': 43.2, 'fri': 45.6,
                  'sat': 44.0}
```



- Accessing an element of dict depends only on its key value
- There is no logical 1st element, 2nd element etc, i.e. unordered

Dictionary - Properties

- Dictionaries are mutable, so we can easily add or remove items
- Unordered
- Have no notion of index position and so cannot be sliced
- **Keys**
  - No duplicate keys allowed (Unique hash val for a key)
  - Keys must be immutable – strings, numbers, tuples
  - If a tuple having any mutable object then it cannot be used as key
- **Values:** have **no restrictions** – Any object, built-in type or user defined – Numbers, strings, lists, sets, dictionaries etc.
- Dict types can be compared using normal **== or !=**
  - <, <=, >, >= are not supported

Dictionary – Points to be noted

- Most flexible data type
- Items are sorted and fetched using “key”, instead of offset
- Highly optimized – Indexing a dictionary element is very fast search operation
- Very useful to represent parse data structures
- **Main Properties:**
    - *Accessed by key, not offset position*
    - *Unordered collections of arbitrary objects*
    - *Variable-length, heterogeneous, and arbitrarily nestable*
    - *Tables of object references (hash tables)*

Create a dict

Method1: Using dict function

```
>>> d1 = dict(one=1, two=2, three=3)
```

```
>>> d2 = dict([('two', 2), ('one', 1), ('three', 3)])
```

```
>>> d3 = dict({'three': 3, 'one': 1, 'two': 2})
```

Method2: Without using dict function

```
>>> d4 = {'one': 1, 'two': 2, 'three': 3}
```

All are equal ?????

```
>>> d1 == d2 == d3 == d4
True
```

Create a dict

Method3: Use expressions and loop

```
>>> d6 = {x: x**2 for x in (1, 2, 3)}
>>> d6
{1: 1, 2: 4, 3: 9}
```

Accessing Values from dict

Printing a value using its key

```
>>> d = dict({'three': 3, 'one': 1, 'two': 2})
>>> d['two']

2

>>> print(d['two'])

2
```

Accessing a value which don’t have a key in dict

```
>>> d = {'one': 1, 'two': 2, 'three': 3}
>>> d['four']
KeyError: 'four'
```

Loop over a dictionary

How to use for loop over a dict?

```
//Create a dict
>>> d = dict({'three': 3, 'one': 1, 'two': 2})
//Print all the keys using loop
>>> for i in d:
       print i
three
two
one
// Print all the values using loop
>>> for i in d:
        print(d[i])
3
1
2
```

Operations on dictionaries

Accessing information about dict

- len(dict)
- get()
- keys()
- values()
- items()
- Iter()
- str()
- in, not in

Modifying a dict

- del
- clear()
- copy()
- pop()
- popitem()

Operations on dictionaries…

(a) **Length of a string – len(d) -** Return the number of items in the dictionary *d*.

```
>>> d = {'one': 1, 'two': 2, 'three': 3}
>>> len(d)
3
```

(b) **Get the value for a given key – d.get(*key*[, *default*])**

```
>>> d.get('two')
2
```

(c) Print all the keys in a dict – d.keys( )

```
>>> d.keys( )
dict_keys(['one', 'three', 'two'])
```

Operations on dictionaries…

(d) d.values() - Return all the values in a dictionary.

```
>>> d.values()
dict_values([1, 3, 2])
```

(e) d.items() - Return a new view of the dictionary"s items ((key, value) pairs).

```
>>> d.items()
dict_items([('one', 1), ('three', 3), ('two', 2)])
```

(f) iter(d) or iter(d.keys()) – Return an iterator over the keys of the dictionary.

```
>>> di = iter(d) //di is key iterator object
<dict_keyiterator object at 0xb70be144>
>>> next(di)
'One'
>>> next(di)
'three'
```

Operations on dictionaries…

(g) str(d) - Produces a printable string representation of a dictionary

```
>>> str(d)
"{'one': 1, 'three': 3, 'two': 2}"
```

(h) Key in d - Return True if d has a key "key", else False.

```
>>> 'one'   in d
True
```

(i) key not in d – Return True if d not have „key", else False.

```
>>> 'one' not in d
False
```

Operations on dictionaries…

(j) del d[key] - Remove d[key] from *d*. Raises a KeyError if *key* is not in the map.

```
>>> del d['one']
>>> d
{'three': 3, 'two': 2}
```

(k) d.clear - Remove all items from the dictionary.

```
>>> d.clear()
>>> d
{ }
```

(l) d.pop(key) – If *key* is in the dictionary d, remove it and return its value, else return *default*.

```
>>> d.pop('one')
>>> d
{'three': 3, 'two': 2}
```

Operations on dictionaries…

(m) popitem[key] - Remove and return **an arbitrary** (key, value) pair from the dictionary.

```
>>> d.popitem[ ]
('one', 1)
>>> d
{'three': 3, 'two': 2}
```

(n) copy(key) – Return a shallow copy of the dictionary. Every key-value tuple in the dictionary is copied. This is not just a new variable reference.

```
>>> dcopy = d.copy()
>>> dcopy
{'one': 1, 'three': 3, 'two': 2}
```

Operations on dictionaries…

(o) d.setdefault(key, <value>) - If *key* is in the dictionary, return its value. If not, insert *key* with a value of *default* and return *default*.

```
>>> d.setdefault('four',4)
4
>>> d.setdefault('five') //Default value is None
None
>>> d
{'four': 4,'one':1,'three': 3, 'five': None, 'two': 2}
```

(p) update(key) – Update the value of a key

```
>>> d.update(five=5)
>>> d
{'four': 4, 'two': 2, 'one': 1, 'three': 3, 'five': 5}
```

Example1 – Most commonly used name

James Govind John

Suresh Ramesh Aravind

Ramesh Ganit Suresh Kapil Suresh

Can we create a dictionary of names ????

Can we find the most commonly used name???

Example1 – Most commonly used name

James **Govind** **John**

Suresh **Ramesh** **Aravind**

Ramesh **Ganit** **Suresh** **Kapil** **Suresh**

Can we create a dictionary of names ????

Can we find the most commonly used name???

Example1 – Most commonly used name

```
# Create a tuple of names
>>>names    =  ('James','Govind',   'John',   'Suresh',
'Ramesh','Aravind',  'Ganit', 'Kapil',        'Suresh',
'Ramesh', 'Suresh')
```

```
# Create an empty dict
>>> countd = dict()
```

```
# Count the frequency of each name, count dictionary
# dict.get(key, default=None) - return value of a key
>>> for i in range(len(names)):
    countd[names[i]] = countd.get(names[i], 0) +  1

# Print the count dictionary
>>> countd
{'Kapil': 1, 'James': 1, 'Aravind': 1, 'Ramesh': 2,
'John': 1, 'Govind': 1, 'Ganit': 1, 'Suresh': 3}
```

Example1 – Most commonly used name…

```
# Find the most commonly used  name - Key with highest
value - Simplest solution
>>> max(count, key=count.get)
'Suresh'
```

```
# Method 2 - using items()
>>> count_new={v:k for k,v in count.items()}
>>> count_new[max(count_new)]
'Suresh'
```

```
# Method 3 - using lambda expression
>>> max(count, key=lambda k: count[k])
'Suresh'
```

Ex1 – Most commonly used name – Using Lists and Dicts

```
# Create a list of names
>>>names    =  ['James','Govind',   'John',   'Suresh',
'Ramesh','Aravind',  'Ganit', 'Kapil',        'Suresh',
'Ramesh', 'Suresh']
```

```
# Create an empty dict
>>> count = dict()
```

```
# Count the frequency of each name
>>>   for name in names :
        count[name] = count.get(name, 0) + 1

# Print the count dictionary
>>> count
{'Kapil': 1, 'James': 1, 'Aravind': 1, 'Ramesh': 2,
'John': 1, 'Govind': 1, 'Ganit': 1, 'Suresh': 3}
```

Example2 – Count the word frequency

Steps:

1. Split the line into words and create a list or tuple
2. Loop through the words
3. Use dict to keep tract of the count of each word

Example2 – Count the word frequency …

```
# Take a line as input and split into words
 >>>line = input('')

# Create an empty dict
>>> Counts = {}

# Split the line into words, travers through all the
words and create a dcitionary of (key, value) = (word,
frequecny)

>>> for word in line.split():
      counts[word] = counts.get(word,0) + 1

>>> counts
{'Is': 2, 'More': 1, 'Tomorrow': 1, 'Then': 1, 'Very':
1, 'The': 1, 'Illusions': 1, 'Today.': 1, 'Life': 1,
'Always': 1, 'Of': 2, 'Time': 1, 'True': 1, 'One': 1,
'Believe': 1, 'That': 1, 'Greatest': 1, 'We': 1,
'There': 1, 'In': 1, 'and': 1}
```

Thank You