DICTIONARIES OPERATION IN PYTHON

Dictionaries

- A **dictionary is like a list,** but more general. In a list, the indices have to be integers; in a dictionary they can be (almost) any type.
- You can think of a dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps to a value. The association of a key and a value is called a **key-value pair** or sometimes an **item.**
- Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair.
- Dictionaries are optimized to retrieve values when the key is known.

Dictionaries

- # empty dictionary
  my_dict = {}

- # dictionary with integer keys
  my_dict = {1: 'apple', 2: 'ball'}

- # dictionary with mixed keys
  my_dict = {'name': 'John', 1: [2, 4, 3]}

- # using dict()
  my_dict = dict({1:'apple', 2:'ball'})

- # from sequence having each item as a pair
  my_dict = dict([(1,'apple'), (2,'ball')])

Nested Dictionary

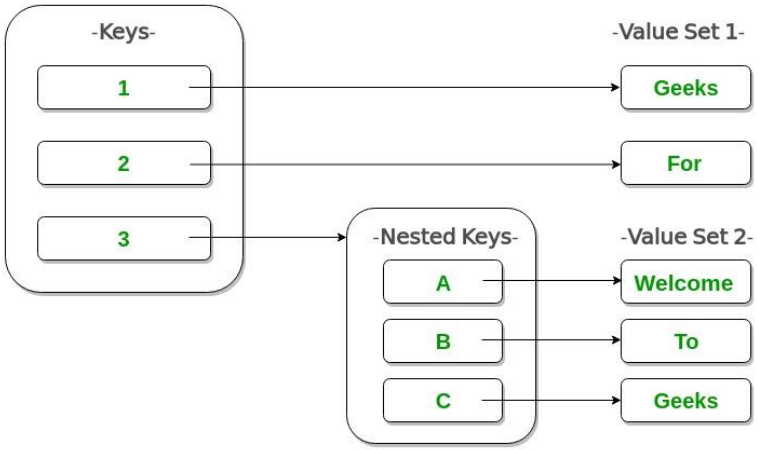
- # Creating a Nested Dictionary
- # as shown in the below image

>>Dict = {1: 'Geeks', 2: 'For', 3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Geeks'}}

>>print(Dict)

>>Dict[3]['B']

>>Dict[1]



Accessing Elements from Dictionary

- # get vs [] for retrieving elements
  my_dict = {'name': 'Jack', 'age': 26}

- # Output: Jack
  print(my_dict['name'])

- # Output: 26
  print(my_dict.get('age'))

```
Jack
26
None
Traceback (most recent call last):
  File "<string>", line 15, in <module>
    print(my_dict['address'])
KeyError: 'address'
```

- # Trying to access keys which doesn't exist throws error
  # Output None
  print(my_dict.get('address'))

- # KeyError
  print(my_dict['address'])

Changing and Adding Dictionary elements

- **Dictionaries are mutable.**
- # Changing and adding Dictionary Elements
  my_dict = {'name': 'Jack', 'age': 26}

- # update value
  my_dict['age'] = 27

- #Output: {'age': 27, 'name': 'Jack'}
  print(my_dict)

```
{'name': 'Jack', 'age': 27}
 {'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

- # add item
  my_dict['address'] = 'Downtown'

- # Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
  print(my_dict)

Removing elements from Dictionary

```
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
print(squares.pop(4)) # remove a particular item, returns its value # Output: 16
print(squares) # Output: {1: 1, 2: 4, 3: 9, 5: 25}

print(squares.popitem()) # remove an arbitrary item, return (key,value) # Output: (5, 25)
print(squares)   # Output: {1: 1, 2: 4, 3: 9}

squares.clear() # remove all items
print(squares) # Output: {}

del squares   # delete the dictionary itself
print(squares)   # Throws Error
```

Type conversion

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack',
    4098)])
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Python Dictionary Methods

Method	Description
clear()	Removes all items from the dictionary.
copy()	Returns a shallow copy of the dictionary.
Fromkeys (seq[, v])	Returns a new dictionary with keys from seq and value equal to v (defaults to None).
get(key[,d])	Returns the value of the key. If the key does not exist, returns d (defaults to None).
items()	Return a new object of the dictionary's items in (key, value) format.
keys()	Returns a new object of the dictionary's keys.
pop(key[,d])	Removes the item with the key and returns its value or d if key is not found. If d is not provided and the key is not found, it raises KeyError.
popitem()	Removes and returns an arbitrary item (key, value). Raises KeyError if the dictionary is empty.
Setdefault (key[,d])	Returns the corresponding value if the key is in the dictionary. If not, inserts the key with a value of d and returns d (defaults to None).
update([other])	Updates the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Returns a new object of the dictionary's values

Python Dictionary Comprehension

- Dictionary comprehension consists of an expression pair (**key: value**) followed by a for statement inside curly braces {}.

```
# Dictionary Comprehension
squares = {x: x*x for x in range(6)}

print(squares)
```

Output

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

This code is equivalent to

```
squares = {}
for x in range(6):
    squares[x] = x*x
print(squares)
```

Output

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Python Dictionary Comprehension

- A dictionary comprehension can optionally contain more for or if statements.
- An optional if statement can filter out items to form the new dictionary.

```
# Dictionary Comprehension with if conditional
odd_squares = {x: x*x for x in range(11) if x % 2 == 1}

print(odd_squares)
```

Output

```
{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
```

Dictionary Membership Test

```
# Membership Test for Dictionary Keys
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True
print(1 in squares)

# Output: True
print(2 not in squares)

# membership tests for key only not value
# Output: False
print(49 in squares)
```

Output

```
True
True
False
```

Iterating Through a Dictionary

```
# Iterating through a Dictionary
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
    print(squares[i])
```

Output

```
1
9
25
49
81
```

Dictionary Built-in Functions

Function	Description
all()	Return True if all keys of the dictionary are True (or if the dictionary is empty).
any()	Return True if any key of the dictionary is true. If the dictionary is empty, return False.
len()	Return the length (the number of items) in the dictionary.
cmp()	Compares items of two dictionaries. (Not available in Python 3)
sorted()	Return a new sorted list of keys in the dictionary.

```
# Dictionary Built-in Functions
squares = {0: 0, 1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: False
print(all(squares))

# Output: True
print(any(squares))

# Output: 6
print(len(squares))

# Output: [0, 1, 3, 5, 7, 9]
print(sorted(squares))
```

Output

```
False
True
6
[0, 1, 3, 5, 7, 9]
```

Histogram (**Dictionary as a set of counters**)

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d
```

The name of the function is **histogram**, which is a statistical term for a set of counters (or frequencies).

```
>>> h = histogram('brontosaurus')
>>> print h
{'a': 1, 'b': 1, 'o': 2, 'n': 1, 's': 2, 'r': 2, 'u': 2, 't': 1}
```

Looping Techniques

items() method

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

Looping Techniques

enumerate() function

When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the enumerate() function.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
 ... print(i, v)
 ...
0 tic
1 tac
2 toe
```

Looping Techniques

zip() function

To loop over two or more sequences at the same time, the entries can be paired with the zip() function.

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

Looping Techniques

reversed() function

```
>>> for i in reversed(range(1, 10, 2)):
...     print(i)
...
9
7
5
3
1
```

. Lexicographical ordering for strings

- Sequence objects typically may be compared to other objects with the same sequence type.
- The comparison uses *lexicographical* ordering: first the first two items are compared, and if they differ this determines the outcome of the comparison; if they are equal, the next two items are compared, and so on, until either sequence is exhausted.
- Some examples of comparisons between sequences of the same type:

Comparing Sequences and Other Types

```
(1, 2, 3)              < (1, 2, 4)
[1, 2, 3]              < [1, 2, 4]
'ABC' < 'C' < 'Pascal' < 'Python'
(1, 2, 3, 4)           < (1, 2, 4)
(1, 2)                 < (1, 2, -1)
(1, 2, 3)              == (1.0, 2.0, 3.0)
(1, 2, ('aa', 'ab'))   < (1, 2, ('abc', 'a'), 4)
```