# 08-Python Basic

Dictionary and dictionary operations

Posted on September 12, 2021
Last updated on February 2, 2023

## 8. Dictionary Data Structure

Dictionaries are used to store data values in `key-value pairs`

- `Duplicate` keys are not allowed but values can be duplicated.
- `Hetrogeneous` objects are allowed for both key and values.
- `Insertion order` is not preserved
- Dictionaries are `mutable`
- `Indexing and slicing` concepts are not applicable

**As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.**

## 8.1 Dictionary declaration

Empty dict declaration

```
p1 = dict()
p2 = {}
print(type(p1), type(p2))
# OP: <class 'dict'> <class 'dict'>
```

Different way of dict declaration

```
d1 = dict([('a',1), ('b',2)])
# OP: {'a': 1, 'b': 2} <class 'dict'>

d2 = dict(a=1, b=2,c=3)
# OP: {'a': 1, 'b': 2, 'c': 3}

d3 = {'a': 1, 3: 'aa'}

person = {}
person['spouse'] = 'Edna'
person['children'] = ['Ralph']
person['pets'] = {'dog': 'Fido'}
# OP: {'spouse': 'Edna', 'children': ['Ralph'], 'pets': {'dog': 'Fidc
```

## 8.2 Keys of Dictionary

Can even use `built-in` objects

```
d4 = {int: 1, float: 2, bool: 3}
# OP: {<class 'int'>: 1, <class 'float'>: 2, <class 'bool'>: 3}
```

Can use `hashable object` but not unhashable object

```
d5 = {'a':1, (1,2): 2}

d6 = {'a':1, (1,2):2, [1,2]:3}
# TypeError: unhashable type: 'list'
```

```
Special char can't be used
```

```
print({234.45 : "abc"})
print({True : "abc"})

print({# : "abc"})
```

## 8.3 Update Dictionary

If the key is not available then a new entry will be added to the dictionary with the specified key-value pair

If the key is already available then old value will be replaced with new value

```
d3['a'] = 2
d3['b'] = 4
```

## 8.4 Display/access Dictionary

```
print(d3['b'])
# 4
print(person['children'])
# ['Ralph']

print(person['children'][0])
# Ralph
```

```
d3 = {'a': 1, 3: 'aa'}
print(d3['c'])
# KeyError: 'c' --> Error if key is absent

print(d3.get('a'))
# None --> if key is absent

print(d3.get('c','NA'))
# NA (takes the default value defined) --> if key is absent
```

## 8.5 Dictionary in-built function

- `items()`
  - It returns list of tuples representing key-value pairs.

```
print(d3.items())
# dict_items([('a', 1), (3, 'aa')])

print(list(d3.items()))
# [('a', 1), (3, 'aa')]
```

- `keys()`
  - It returns all keys associated eith dictionary

```
print(d3.keys())
print(list(d3.keys()))
```

- `values()`
  - It returns all values associated with the dictionary

```
print(d3.values())
print(list(d3.values()))
print()
```

- `pop()`
  - It removes the entry associated with the specified key and returns the corresponding value

```
d2 = {'a': 1, 'b': 2, 'c': 3}

d2.pop('c')
# 3 --> poped: values

d2.pop('c')
# KeyError: 'c'  --> Error if key is absent

# takes the default value defined --> if key is absent
d2.pop('d','NA')
# 'NA'
```

- `popitem()`

- It removes an arbitrary item(key-value) from the dictionaty and returns it.

```
d2.popitem()
# ('b', 2)  # (key, value)
```

- `update()`
  - All items present in the dictionary x will be added to dictionary d

```
d1 = {'b': 200, 'd': 400}
d2 = {'a': 1, 'c': 3}

d1.update(d2)
# d1 OP: {'b': 200, 'd': 400, 'a': 1, 'c': 3}

d2.update([('y', 2), ('z', 4)])
# d2 OP: {'a': 1, 'c': 3, 'y': 2, 'z': 4}

d2.update(p=22, q=44)
# d2 OP: {'a': 1, 'c': 3, 'y': 2, 'z': 4, 'p': 22, 'q': 44}
```

- `clear()`
  - To remove all entries from the dictionary

```
d2.clear()
# OP: {}
```

- `setdefault()`
  - The setdefault() method returns the value of a key (if the key is in dictionary).
  - If not, it inserts key with a value to the dictionary.

```
person = {'name': 'Phill', 'age': 22}

# ignore new value and return old value
print(person.setdefault('age','30')) # 22

# add new value and return
print(person.setdefault('dbo','1994')) # 1994


# working with list --> no return value
# if no 'city' key, then create empty list and append, else just appe
person.setdefault('city', []).append('ghy')
```

## 8.6 Dictionary Comprehension

```
dict1 = {i:i ** 3 for i in range(8) if i % 2 != 0}
print(dict1)
# {1: 1, 3: 27, 5: 125, 7: 343}

state = ['Gujarat', 'Maharashtra', 'Rajasthan']
capital = ['Gandhinagar', 'Mumbai', 'Jaipur']
dict1 = {key:value for (key, value) in zip(state, capital)}
print(dict1)

# {'Maharashtra': 'Mumbai', 'Rajasthan': 'Jaipur',
  'Gujarat': 'Gandhinagar'}
```

## 8.7 Iterate Through a Dictionary

```
a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
```

By default it will access `only keys`

```
for key in a_dict:
  print(key, ' --> ', a_dict[key])

# color  -->  blue
# fruit  -->  apple
# pet  -->  dog
```

Access keys

```
for key in a_dict.keys():
  print(key)

# color
# fruit
# pet
```

Access values

```
for value in a_dict.values():
  print(value)

# blue
# apple
# dog
```

Access key-value pair

```
for item in a_dict.items():
        print(item)

# ('color', 'blue')
# ('fruit', 'apple')
# ('pet', 'dog')
```

Access key-value pair

```
for key, value in a_dict.items():
        print(key, '-->', value)

# color  -->  blue
# fruit  -->  apple
# pet  -->  dog
```

Sort by keys

```
for key in sorted(a_dict):
  print(key, ' --> ', a_dict[key])
```

Sort by values

```
def by_value(item):
  return item[1]

for k, v in sorted(a_dict.items(), key=by_value):
  print(k, ' --> ', v)
```

Get only sorted values

```
for value in sorted(a_dict.values()):
  print(value)
```

## Usage

```
prices = {'apple': 0.40, 'orange': 0.35, 'banana': 0.25}

# for key in prices.keys():
# RuntimeError: dictionary changed size during iteration
# So, list is used
for key in list(prices.keys()):
  if key == 'orange':
    del prices[key]
print(prices)
# {'apple': 0.4, 'banana': 0.25}
```

.

.

.

.

.

.

.

.

.

.

.

.

.