ECE326

PROGRAMMING LANGUAGES

Lecture 4a : Unordered Types in Python

Kuei (Jack) Sun
ECE
University of Toronto
Fall 2020

Dictionary

- As known as associative array
  - std::unordered_map in C++
- Collection of key value pairs
  - All keys must be unique
  - Unordered
    - You cannot sort a dictionary
- Implementation
  - Hash table
  - Search tree (e.g. red-black tree)

Dictionary

- { key1:value1, key2:value2, … }

```
>> eng2sp = {‘one’: ‘uno’, ‘two’: ‘dos’, ‘three’: ‘tres’}
>> eng2sp['one']
'uno'
>> eng2sp[4]                          # key not in dictionary
KeyError: 4

>> eng2sp['four'] = 'cuatro’          # add key-value pair
>> eng2sp
{'one': 'uno', 'two': 'dos', 'three': 'tres', 'four': 'cuatro'}

>> 'two' in eng2sp      # membership test on key
True
>> ‘dos’ in eng2sp      # cannot use to check if value exists
False
```

Key Requirement

- Dictionary key must be *hashable*
  - Must be *immutable*
  - Must not have references to *mutable* objects

```
>> d = dict()      # creates an empty dictionary
>> d[1,2] = “hi”   # OK
>> a = [1, 2]
>> d[a] = “bye”    # not OK, list is mutable
TypeError: unhashable type: 'list‘
>> t = (1, a)
>> t
(1, [1, 2])
>> d[t] = “bye”    # not OK, tuple contains a mutable object
TypeError: unhashable type: 'list‘
```

Building Dictionary

- Problem
  - 2 lists with equal length
  - Want to make a dictionary

```
eng = ["one", "two", "three"]
jap = ["ichi", "ni", "san"]

e2j = dict()
for i in range(len(eng)):
    e2j[eng[i]] = jap[i]

print(e2j)
# {'one': 'ichi', 'two': 'ni', 'three': 'san'}
```

Building Dictionary

- `zip` built-in function
  - Creates *n* *m*-tuples from *m* sequences of length *n*
    - Each element in tuple taken from same position of each sequence
    - *Lazy iterable*

```
>> a = zip(“abcd”, range(4), [1.5, 2.5, 3.5, 4.5])
>> a                          # zip is ‘lazy’
<zip object at 0x6ffffcc0888>
>> list(a)                    # consume the iterable
[('a', 0, 1.5), ('b', 1, 2.5), ('c', 2, 3.5), ('d', 3, 4.5)]
```

- Build dictionary with list of 2-tuples (key-value pairs)

```
>> dict(zip('abcde', range(5)))
{'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4}
```

Dictionary Methods

```
d = dict(zip('abcde', range(5)))

>> for c in d:                    # loops through key only
..     print(c, end="")
abced

>> d.keys()
dict_keys(['a', 'b', 'c', ‘e', ‘d’])
>> for c in sorted(d.keys(), reverse=True):
..     print(d[c], end="")
43210

>> d.values()
dict_values([0, 1, 2, 3, 4])

>> for k, v in d.items():         # loops through (key, value)
..     print(k+str(v), end=' ‘)
a0 b1 c2 d3 e4
```

Dictionary Methods

```
>> d = dict(zip('abcdef', range(6)))

>> del d['a’]                # remove key ‘a’ and its value
>> d
{'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f’: 5}

>> d.pop('c’)                # same as above and return its value
2
>> d
{'b': 1, 'd': 3, 'e': 4, 'f’: 5}

>> d.update(dict(zip('abc', range(6,9)))) # ‘b’ gets new value
>> d
{'b': 7, 'd': 3, 'e': 4, 'f': 5, 'a': 6, 'c': 8}
```

Set

- Same as list, but cannot have duplicate elements
  - std::unordered_set in C++
- Syntax: `{ value1, value2, … }`

```
a = { 2, 5, "hello" }
```

- Programming Idiom
  - A language-specific convention of accomplishing a task
  - E.g. removing duplicates

```
old = [3, 2, 3, 1, 6, 5, 1, 3, 9, 6]
new = []
for i in old:
    if i not in new:
        new.append(i)
```

```
>> new = list(set(old))
>> new
[1, 2, 3, 5, 6, 9]
```

Set Methods

- No subscript operator
  - Can only loop through it

```
d = set(range(6))

>> d.remove(3)
>> d
{0, 1, 2, 4, 5}

>> st.add(9)
>> st
{0, 1, 2, 4, 5, 9}
```

```
>> st | set(range(4,7)) # union of 2 sets
{0, 1, 2, 4, 5, 6, 9}

>> st & set(range(4, 7)) # intersect
{4, 5}

>> st ^ set(range(4, 7)) # difference
{0, 1, 2, 6, 9}
```