Data Structures

Built-in Dict and Set

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Built-in **Dict and Set**

- The course prerequisite was already to know these 2 data structures
- Internally, they are built with hash table
  - The main difference, the key in set is not attached to a value
- We already implemented our variants of it
- Think about its constraints, such as immutability for keys and their relationships with the hash function
- There is also OrderedDict
  - The order of the inserted keys is remembered
  - Keys order NOT keys values
- In general, hash table is good if you DON’T care about items order
  - E.g. you don’t care items are sorted
  - E.g. when we use dict in a trie, we don’t care about items order

```python
# immutables: int, float, tuple, string
# be careful from float as it is an approximate - don't

class Employee:
    pass

dict = {
    -1200001 : 'mostafa',
    'ziad' : 25.5,
    (4, 6) : [5, 8, 9],
    'Hey' : Employee(),
    16 : {6:90},     # value is another dict
    int: [6, 9, 10],
    #[1, 2] : 10    TypeError: unhashable type: 'list'
    #([1, 2]) : 10   TypeError: unhashable type: 'list'
}
print(dict[(4, 6)]) # [5, 8, 9]

st = set()
st.add(10)
st.add(10)
st.add(10)

print(len(st))      # 1
```

Recall

```
class PhoneEntry:
    def __init__(self, name, number, address):
        self.name = name
        self.number = number
        self.address = address

    def __hash__(self):
        tup = (self.name, self.number)
        return hash(tup)


    def __eq__(self, other):
        return self.name == other.name and \
               self.number == other.number
```

Immutability matters

```
p1 = PhoneEntry('Most', '123', 'Egypt')
p2 = PhoneEntry('Most', '123', 'Canada')

dct = {}
dct[p1] = 90
dct[p2] = 80

print(dct[p1])  # 80
print(dct[p2])  # 80
print(len(dct)) # 1
p1.name = 'another'
#print(dct[p1])  # 90   KeyError
# Don't change the object once entered in the hash table
# It will just make unexpected errors
# That is why immutability matter
```

Hash/Equality consistency matters

```
class PhoneEntry:
    def __init__(self, name, number, address):
        self.name = name
        self.number = number
        self.address = address

    def __hash__(self):
        return hash((self.name, self.number))

    def __eq__(self, other):
        return False

p1 = PhoneEntry('Most', '123', 'Egypt')
p2 = PhoneEntry('Most', '123', 'Canada')
dct = {}
dct[p1] = 90
dct[p2] = 80
print(len(dct))     # 2
```

- When 2 objects have the same hash, then we use the equality to compare their objects
- For the hash table:
  - Different hash ⇒ Must be different objects
  - 2 objects are equal IFF
    - Same hash and Equal Comparison
- For sorting:
  - 2 objects are compared based on:
    - __cmp__, __ge__, __le__
    - *Hash is ignored*
- Be consistent to not waste days in debugging
  - Equal objects? Must be same hash

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."