Hash maps

→ like Python's **dict**, order is not-guaranteed

Create

```
dict_ = dict()
dict_['a'], dict_['b'], dict_['c'] = 1, 2, 3
dict_ = dict([('a', 1), ('b', 2), ('c', 3)])
dict_ = dict(a=1, b=2, c=3)
dict_ = {x[0]: int(x[1]) for x in 'a1 b2 c3'.split()}
dict_ = {'a':1, 'b':2, 'c':3}
dict_
>> {'a': 1, 'b': 2, 'c': 3}
```

Read

```
dict_['b']                  >> 2
dict_.get('d', 99)          >> 99
dict_                       >> {'a': 1, 'b': 2, 'c': 3}
dict_.setdefault('c', 33)   >> 3
dict_                       >> {'a': 1, 'b': 2, 'c': 3}
dict_.setdefault('d', 44)   >> 44
dict_                       >> {'a': 1, 'b': 2, 'c': 3, 'd': 44}
dict_.keys()                >> dict_keys(['a', 'b', 'c', 'd'])
```

Update

```
dict_['b'] = 22
dict_                       >> {'a': 1, 'b': 22, 'c': 3, 'd': 44}
```

Delete

```
dict_.pop('d')              >> 44
dict_                       >> {'a': 1, 'b': 22, 'c': 3}
```

Loop

```
for item in dict_.items():
    print(item)
>>
('a', 1)
('b', 22)
('c', 3)

for k, v in dict_.items():
    print(k, v)
>>
a 1
b 22
c 3

#for item in dict_:
for item in dict_.keys():
    print(item)
>>
a
b
c
```

Membership testing / Others:

```
dict_              >> {'a': 1, 'b': 22, 'c': 3}
'a' in dict_       >> True
'z' in dict_       >> False
len(dict_)         >> 3
```

Sets

→ order is not-guaranteed

Create

```
set_ = set()
set_.add('a'), set_.add('b'), set_.add('c'), set_.add('a')
set_ = set('a b c a'.split())
set_ = set('abca') # NOT set('abca'.split())
set_ = {x for x in 'abc'}
set_ = {'a', 'b', 'c', 'a'}
set_
>>
{'a', 'b', 'c'}
```

Delete

```
set_.remove('b')      >>
set_                  >> {'a', 'c'}
```

Loop

```
for item in set_:
    print(item)
>>
c
a
```

Set operations

```
s1 = set('abcd')
s1                    >> {'a', 'b', 'c', 'd'}
s2 = set('cdef')
s2                    >> {'c', 'd', 'e', 'f'}
s1 - s2               >> {'a', 'b'}
s1 | s2               >> {'a', 'b', 'c', 'd', 'e', 'f'}
s1 & s2               >> {'c', 'd'}
s1 ^ s2               >> {'a', 'b', 'e', 'f'}
```

Membership testing / Others:

```
set_              >> {'a', 'b', 'c'}
'a' in set_       >> True
'z' in set_       >> False
len(set_)         >> 3
```

Stack → List

Adapter design pattern

```
S.push(e)       → L.append(e)
S.pop()         → L.pop()
S.top()         → L[-1]
S.is_empty()    → len(L) == 0
len(S)          → len(L)
```

Queue → List

Use **circular list** instead!

The modulo operator % is ideal for circular operations.

```
Q.enqueue(e)    → L.append(e)
Q.dequeue()     → L.pop(0) inefficient!
Q.first()       → L[0]
Q.is_empty()    → len(L) == 0
len(Q)          → len(L)
```