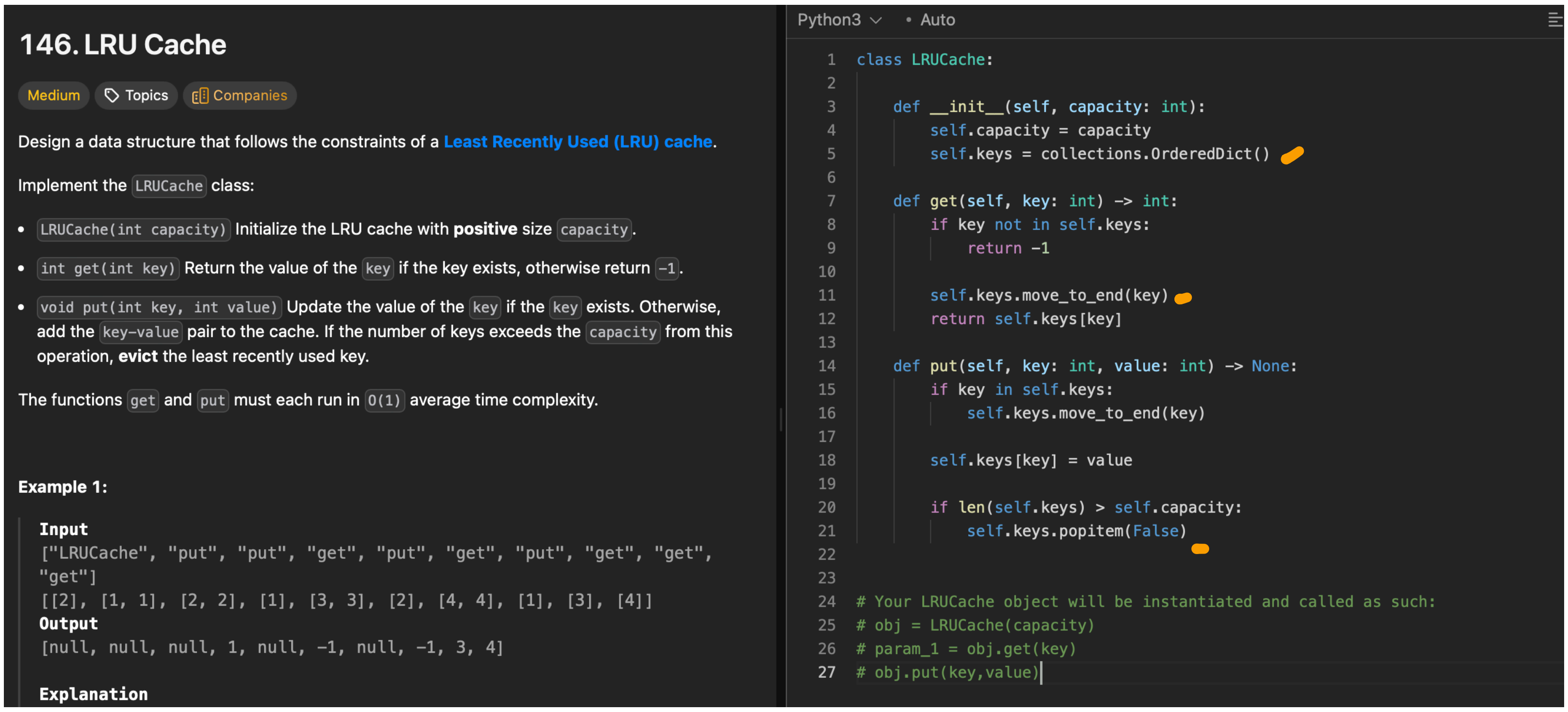


146. LRU Cache

Medium | Topics | Companies

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `-1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

The functions `get` and `put` must each run in `O(1)` average time complexity.

Example 1:

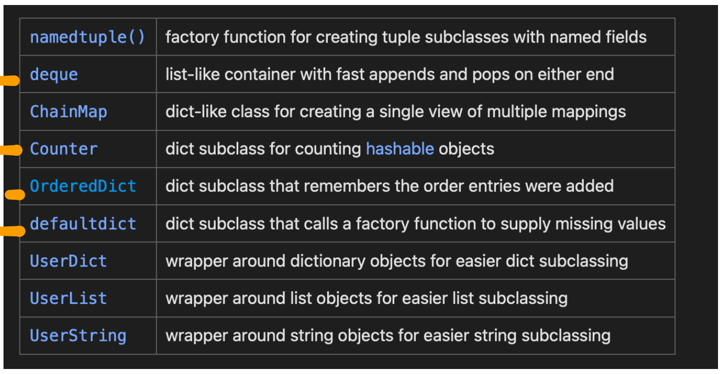
```
Input
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation
```

Python3 · Auto

```python
class LRUCache:

    def __init__(self, capacity: int):
        self.capacity = capacity
        self.keys = collections.OrderedDict()

    def get(self, key: int) -> int:
        if key not in self.keys:
            return -1

        self.keys.move_to_end(key)
        return self.keys[key]

    def put(self, key: int, value: int) -> None:
        if key in self.keys:
            self.keys.move_to_end(key)

        self.keys[key] = value

        if len(self.keys) > self.capacity:
            self.keys.popitem(False)


# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```

collections.OrderedDict()

https://docs.python.org/3/library/collections.html#

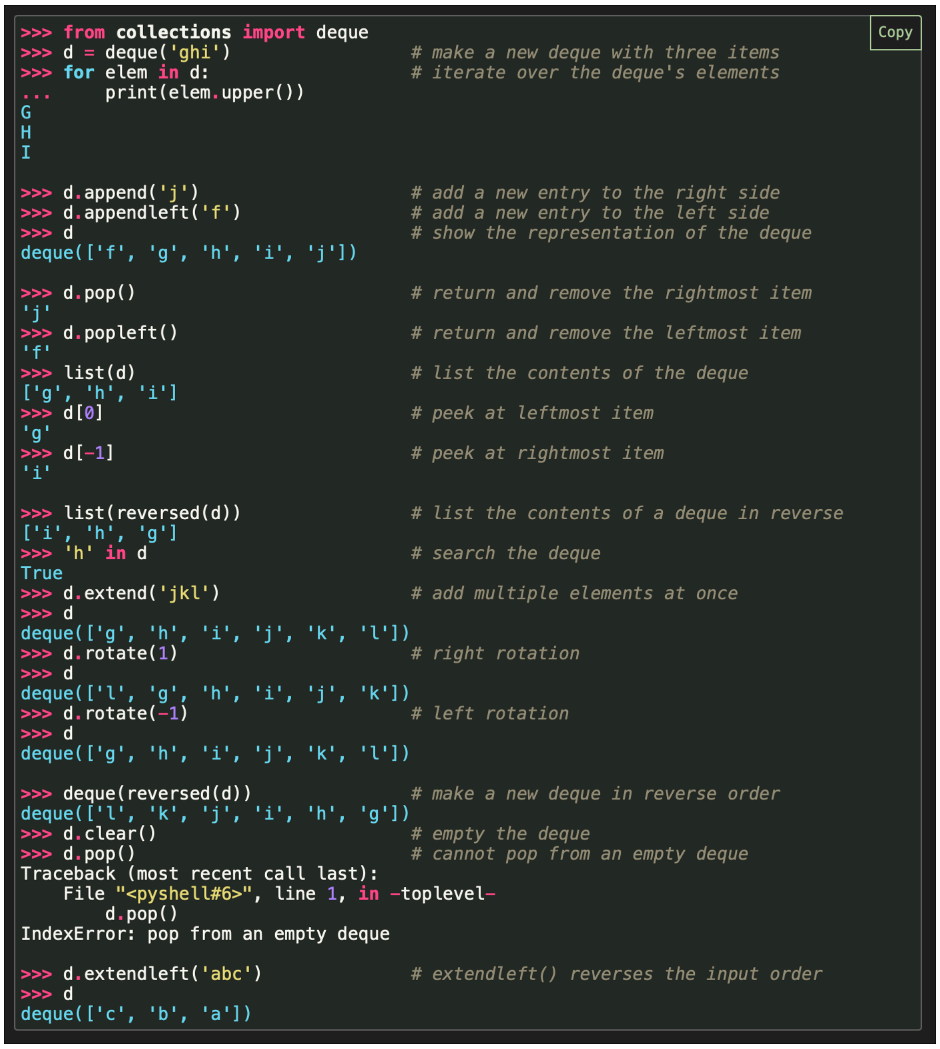


`namedtuple()`	factory function for creating tuple subclasses with named fields
`deque`	list-like container with fast appends and pops on either end
`ChainMap`	dict-like class for creating a single view of multiple mappings
`Counter`	dict subclass for counting hashable objects
`OrderedDict`	dict subclass that remembers the order entries were added
`defaultdict`	dict subclass that calls a factory function to supply missing values
`UserDict`	wrapper around dictionary objects for easier dict subclassing
`UserList`	wrapper around list objects for easier list subclassing
`UserString`	wrapper around string objects for easier string subclassing

these four container datatypes are most frequent used
在implement时候可以省略前面的 collections 就像Counter， deque直接用

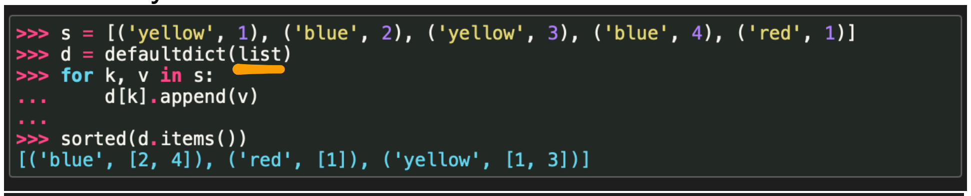
- deque
  - class collections.deque([iterable [, maxlen]])
    - Returns a new deque object initialized left-to-right (using append()) with data from iterable. If iterable is not specified, the new deque is empty. If maxlen is not specified or is None, deques may grow to an arbitrary length.
  - methods:
    - append(x)
    - appendleft(x)
    - clear()
    - copy()
    - count(x)
    - extend(iterable)
    - extendleft(iterable)
    - index(x[, start[, stop]])
    - insert(i, x)
    - pop()

- popleft()
- remove(val)
- reverse()
- rotate(n=1): Rotate the deque n steps to the right. If n is negative, rotate to the left. When the deque is not empty, rotating one step to the right is equivalent to d.appendleft(d.pop()), and rotating one step to the left is equivalent to d.append(d.popleft()). rotate(n) 等于把后n个放到前面来
- maxlen

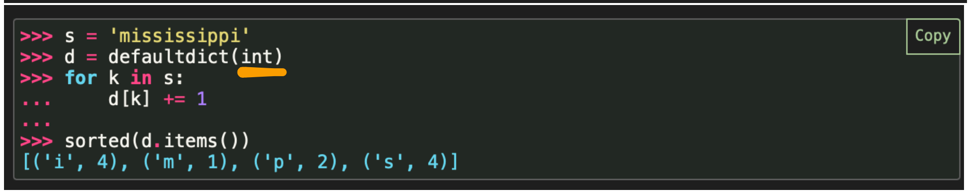


```
>>> from collections import deque
>>> d = deque('ghi')                 # make a new deque with three items
>>> for elem in d:                   # iterate over the deque's elements
...     print(elem.upper())
G
H
I

>>> d.append('j')                    # add a new entry to the right side
>>> d.appendleft('f')                # add a new entry to the left side
>>> d                                # show the representation of the deque
deque(['f', 'g', 'h', 'i', 'j'])

>>> d.pop()                          # return and remove the rightmost item
'j'
>>> d.popleft()                      # return and remove the leftmost item
'f'
>>> list(d)                          # list the contents of the deque
['g', 'h', 'i']
>>> d[0]                             # peek at leftmost item
'g'
>>> d[-1]                            # peek at rightmost item
'i'

>>> list(reversed(d))                # list the contents of a deque in reverse
['i', 'h', 'g']
>>> 'h' in d                         # search the deque
True
>>> d.extend('jkl')                  # add multiple elements at once
>>> d
deque(['g', 'h', 'i', 'j', 'k', 'l'])
>>> d.rotate(1)                      # right rotation
>>> d
deque(['l', 'g', 'h', 'i', 'j', 'k'])
>>> d.rotate(-1)                     # left rotation
>>> d
deque(['g', 'h', 'i', 'j', 'k', 'l'])

>>> deque(reversed(d))               # make a new deque in reverse order
deque(['l', 'k', 'j', 'i', 'h', 'g'])
>>> d.clear()                        # empty the deque
>>> d.pop()                          # cannot pop from an empty deque
Traceback (most recent call last):
    File "<pyshell#6>", line 1, in -toplevel-
        d.pop()
IndexError: pop from an empty deque

>>> d.extendleft('abc')              # extendleft() reverses the input order
>>> d
deque(['c', 'b', 'a'])
```

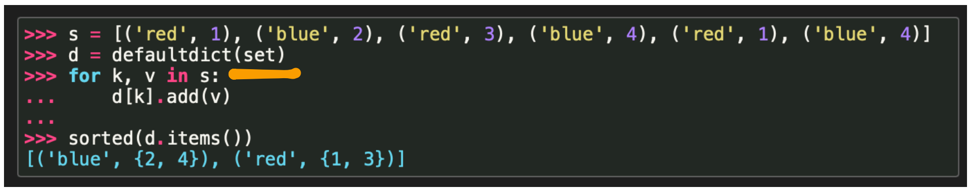
- defualtdict()
    - Using list as the default_factory, it is easy to group a sequence of key-value pairs into a dictionary of lists



```
>>> s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
>>> d = defaultdict(list)
>>> for k, v in s:
...     d[k].append(v)
...
>>> sorted(d.items())
[('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
```

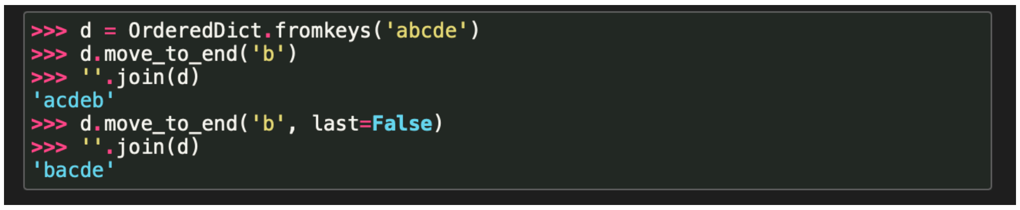


```
>>> s = 'mississippi'
>>> d = defaultdict(int)
>>> for k in s:
...     d[k] += 1
...
>>> sorted(d.items())
[('i', 4), ('m', 1), ('p', 2), ('s', 4)]
```



```
>>> s = [('red', 1), ('blue', 2), ('red', 3), ('blue', 4), ('red', 1), ('blue', 4)]
>>> d = defaultdict(set)
>>> for k, v in s:
...     d[k].add(v)
...
>>> sorted(d.items())
[('blue', {2, 4}), ('red', {1, 3})]
```

- OrderedDict
    - class collections.OrderedDict([items])
    - They have become less important now that the built-in dict class gained the ability to remember insertion order
    - method:
        - move_to_end(key, last=True): The item is moved to the right end if last is true (the default) or to the beginning if last is false, raises KeyError if the key does not exist



```
>>> d = OrderedDict.fromkeys('abcde')
>>> d.move_to_end('b')
>>> ''.join(d)
'acdeb'
>>> d.move_to_end('b', last=False)
>>> ''.join(d)
'bacde'
```

        - popitem(last=True) : LIFO order if last is true or FIFO order if false.