Data structures

Python Reference

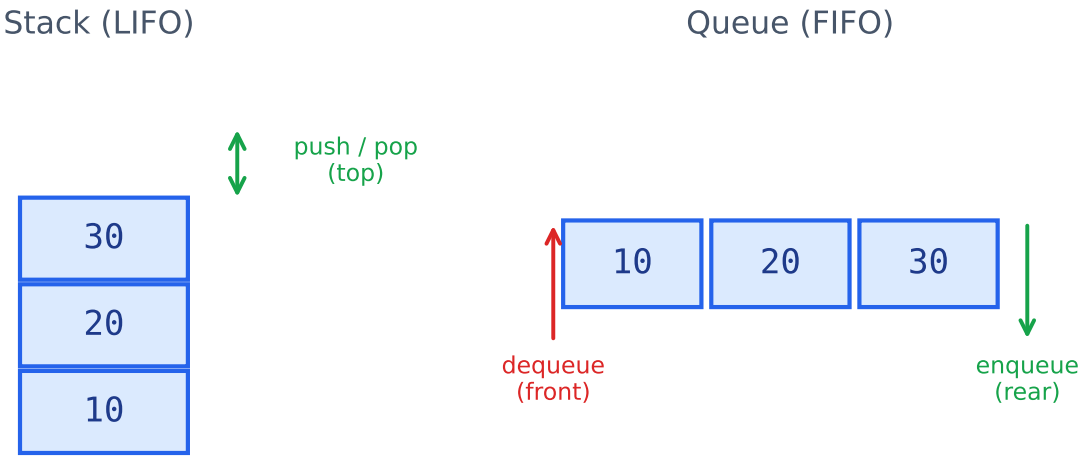
Abstract Data Types (ADTs)

An abstract data type 抽象数据类型 (ADT) describes some data plus the operations on it, separate from how it is built. You use it through its operations, not through its inner storage.

```
# A stack ADT, built on a list
s = []
s.append(1)        # add
s.append(2)
print(s.pop())     # 2  (remove the most recent)
```

Stacks

A stack 栈 is last-in, first-out (LIFO 后进先出). You push 压入 onto the top and pop 弹出 from the top.



A stack removes from the top (LIFO); a queue removes from the front (FIFO)

```
stack = []
stack.append("a")
stack.append("b")
print(stack.pop())   # b
print(stack.pop())   # a
```

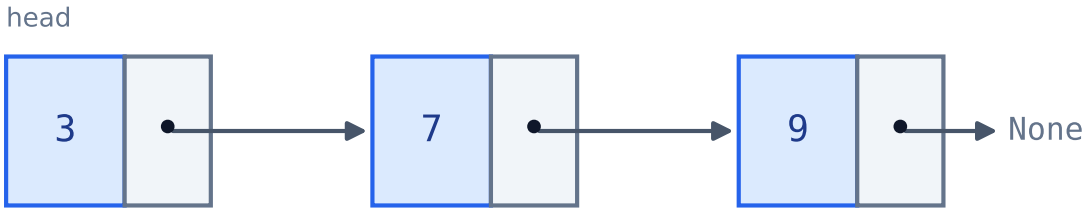
Queues

A queue 队列 is first-in, first-out (FIFO 先进先出). You enqueue 入队 at the back and dequeue 出队 from the front.

```
queue = []
queue.append("a")     # enqueue
queue.append("b")
print(queue.pop(0))   # a  (dequeue the front)
print(queue.pop(0))   # b
```

Linked lists

A linked list 链表 is a chain of nodes 节点. Each node holds data and a pointer 指针 to the next node; the last points to `None`.

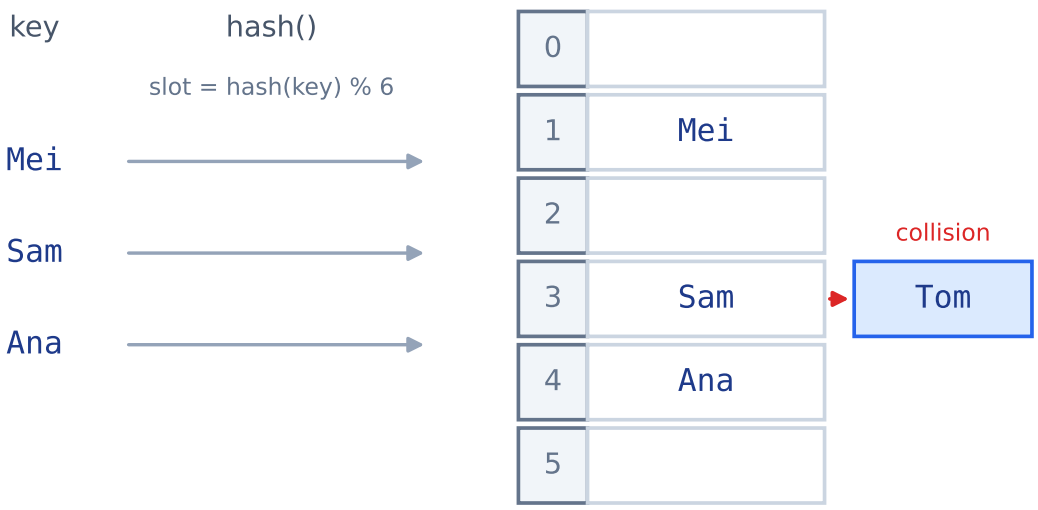


A linked list: each node holds data and a pointer to the next node, ending at None

```
n3 = {"data": 3, "next": None}
n2 = {"data": 2, "next": n3}
n1 = {"data": 1, "next": n2}
node = n1
while node is not None:      # traverse to the end
    print(node["data"])
    node = node["next"]
# 1 2 3
```

Hash tables

A hash table 哈希表 maps a key to a slot with a hash function 哈希函数. Two keys can land in the same slot —a collision 冲突. Python's `dict` is a hash table, so lookup is fast.

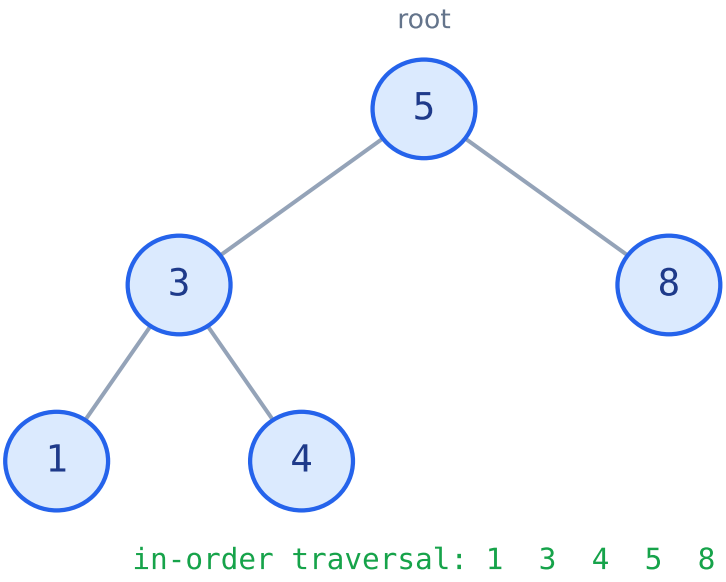


A hash function maps each key to a slot; two keys can collide in one slot

```
table = {}
table["Mei"] = 88
table["Sam"] = 71
print(table["Mei"])   # 88  (fast lookup by key)
```

Binary search trees

A binary search tree 二叉搜索树 (BST) keeps order: every left child is smaller than its node, every right child is larger. Search stays fast.



A binary search tree: smaller values go left, larger values go right

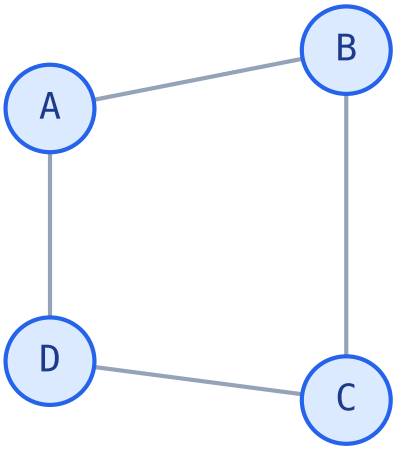
```python
def insert(root, val):
    if root is None:
        return {"val": val, "left": None, "right": None}
    if val < root["val"]:
        root["left"] = insert(root["left"], val)
    else:
        root["right"] = insert(root["right"], val)
    return root

def inorder(root):
    if root is None:
        return []
    return inorder(root["left"]) + [root["val"]] + inorder(root["right"])

tree = None
for v in [5, 3, 8, 1, 4]:
    tree = insert(tree, v)
print(inorder(tree))   # [1, 3, 4, 5, 8]
```

Graphs

A graph 图 is a set of vertices 顶点 joined by edges 边. An adjacency list 邻接表—a dict of neighbour lists —is a common way to store one.




```python
graph = {
  "A": ["B", "D"],
  "B": ["A", "C"],
  "C": ["B", "D"],
  "D": ["A", "C"],
}
```

A graph of vertices and edges, with its adjacency-list form

```python
graph = {"A": ["B", "D"], "B": ["A", "C"], "C": ["B", "D"], "D": ["A",
↪  "C"]}
for vertex in graph:
    print(vertex, "->", graph[vertex])
# A -> ['B', 'D']  (and so on for B, C, D)
```