Stack and Queues

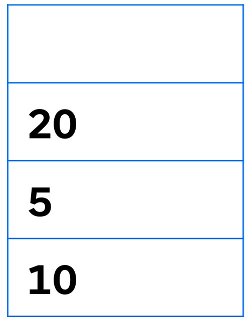
Agenda

Review

- Stacks
- Queues

Stacks and Queues

Stack: Linear data structure with LIFO (Last In, First Out)



Last in first out (LIFO)

Stacks

Key Operations:

1. Push: Add to the top.
2. Pop: Remove from the top.
3. Peek/Top: View the top.
4. IsEmpty: Check if empty.

Use Cases: Undo operations, Syntax parsing, DFS.

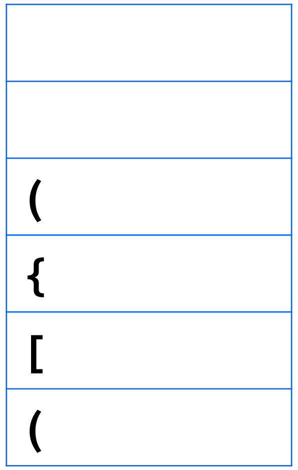
Classic Stack Problem

Bracket Matching: Determine if bracket sequence is valid.

Example: ([{()}]{})

Iterate through string.

- Push opening brackets onto stack.
- If encounter a closing bracket, pop top element and continue if it is a matching opening bracket. If not, the sequence is invalid.



Don’t push). Pop top element and check if it’s the matching opening. Continue

Push (

Push {

Push [

Push (

Queues

Linear data structure with FIFO (First In, First Out)

10	**5**	**20**	

First in first out (FIFO)

Queues

Key Operations:

1. Enqueue: Add to the rear.
2. Dequeue: Remove from the front.
3. Peek/Front: View the front.
4. IsEmpty: Check if empty.

Use Cases: Task scheduling, BFS, Web servers.

Priority Queue

Element with highest priority is first element out

Implement with a heap

10	**25**	**20**	

First in first out (FIFO)

Using Python’s heapq

Python’s heapq can be used to implement a priority queue

Heapq’s default behavior is a **min** heap meaning the minimum element in the heap will always be the root as opposed to the maximum element as the root.

```
import heapq
pq = []  # Empty priority queue (min-heap)
heapq.heappush(pq, 5)  # Insert 5
heapq.heappush(pq, 2)  # Insert 2
heapq.heappush(pq, 8)  # Insert 8

print(pq)  # Output: [2, 5, 8] (Smallest element at index 0)
smallest = heapq.heappop(pq)  # Removes and returns the smallest element
print(smallest)  # Output: 2
print(pq)  # Output: [5, 8]
print(pq[0])  # Output: 5 (smallest element)
```

heapq and max heap

To use heapq as a **max** heap, prior to Python 3.14 a workaround was to negate the values that are entered into the heap.

```
max_pq = []
# Add 5,2,8 to the heap
heapq.heappush(max_pq, -5)  # Push negative values
heapq.heappush(max_pq, -2)
heapq.heappush(max_pq, -8)

print(-heapq.heappop(max_pq))  # Output: 8 (largest element)
```

heapq and max heap

Python 3.14 introduced functions for a **max** heap.

```
import heapq
pq = []  # Empty priority queue (min-heap)
heapq.heappush_max(pq, 5)  # Insert 5
heapq.heappush_max(pq, 2)  # Insert 2
heapq.heappush_max(pq, 8)  # Insert 8

print(pq)  # Output: [8, 2, 5] (Smallest element at index 0)
largest = heapq.heappop_max(pq)  # Removes and returns the smallest element
print(largest)  # Output: 8
print(pq)  # Output: [5, 2]
print(pq[0])  # Output: 5 (largest element)
```