CHATGPT CHAT

🔤 Strings

Function	Use Case	Example
str[::-1]	Reverse a string	"abc"[::-1] → "cba"
str.split(delim)	Split string by delimiter	"a,b,c".split(',') → ['a', 'b', 'c']
str.join(list)	Join list into a string	'-'.join(['a','b']) → "a-b"
str.find(sub)	Find index of substring	"abcde".find("cd") → 2
str.replace(old, new)	Replace part of string	"abc".replace('b','x') → "axc"
str.isalnum() / isalpha() / isdigit()	Character classification	"abc123".isalnum() → True
ord(c) / chr(n)	Char to int and vice versa	ord('a') → 97, chr(97) → 'a'
collections.Counter()	Count frequency of characters	Counter("aab") → {'a':2,'b':1}

📚 Lists / Arrays

Function	Use Case	Example
sum(lst)	Sum of all elements	sum([1,2,3]) → 6
max(lst) / min(lst)	Max/Min value	max([1,2,3]) → 3
sorted(lst)	Sort a list (returns new list)	sorted([3,1,2]) → [1,2,3]
list.sort()	In-place sort	lst.sort()
reversed(lst)	Reverse iterator	list(reversed([1,2,3])) → [3,2,1]
enumerate(lst)	Get index and value	for i, val in enumerate(lst)
zip(a, b)	Pair elements	zip([1,2],[3,4]) → [(1,3),(2,4)]
any() / all()	Logical checks	any([False, True]) → True
itertools.permutations()	All permutations	permutations([1,2]) → (1,2),(2,1)
itertools.combinations()	All combinations	combinations([1,2,3],2) → (1,2),(1,3),(2,3)

🔁 Loops and Functional Tools

Function	Use Case	Example
map(func, iterable)	Apply function to all items	map(str, [1,2]) → ['1','2']
filter(func, iterable)	Filter items	filter(lambda x: x>0, [-1,2]) → [2]
lambda	Anonymous function	lambda x: x*2

🔢 Math / Numbers

Function	Use Case	Example
abs(x)	Absolute value	abs(-4) → 4
pow(x, y, mod)	Exponentiation (modulo optional)	pow(2,3) → 8
divmod(a, b)	Get quotient & remainder	divmod(7,3) → (2,1)
math.gcd() / math.lcm()	GCD/LCM	gcd(8,12) → 4
math.sqrt()	Square root	sqrt(16) → 4.0
math.log(x, base)	Logarithm	log(8, 2) → 3

📦 Dictionaries / HashMaps

Function	Use Case	Example
dict.get(key, default)	Safe access	d.get('x', 0)
defaultdict(list/int)	Auto-create keys	d = defaultdict(list)
collections.Counter()	Frequency counting	Counter([1,2,2,3]) → {2:2, 1:1, 3:1}

🧮 Sets

Function	Use Case	Example
set()	Create unique collection	set([1,2,2]) → {1,2}
set1 & set2	Intersection	{1,2} & {2,3} → {2}
`set1	set2`	Union	
set1 - set2	Difference	{1,2} - {2} → {1}

📚 Heaps / Priority Queues

Function	Use Case	Example
heapq.heappush(heap, item)	Push to heap	
heapq.heappop(heap)	Pop smallest item	
heapq.heapify(lst)	Make heap from list	
heapq.nlargest(n, iterable)	Get n largest	
heapq.nsmallest(n, iterable)	Get n smallest	

📝 Deque (Double-Ended Queue)

From collections:

Function	Use Case	Example
append() / appendleft()	Add to right/left	
pop() / popleft()	Remove from right/left	
deque([..])	Sliding window, BFS	

📐 Useful Algorithms

Function / Tool	Use Case
two pointers	Palindrome check, sorted arrays
sliding window	Longest substring, subarray sum
prefix sum	Range sum queries
binary search (bisect)	Search in sorted arrays
DFS / BFS (with deque)	Graph traversal

📌 Real Examples in LeetCode Problems

Problem Type	Function that Helps
Palindrome check	s == s[::-1]
Anagram	Counter(s1) == Counter(s2)

Problem Type	Function that Helps
Max in sliding window	heapq + deque
K closest points	heapq.nsmallest()
Two sum	dict.get() for lookup
Substring problems	set() + sliding window
Frequency sort	Counter + sorted(..., key=...)

🔤 STRING FUNCTIONS

Function	Example	Use	Output	Problem
s[::-1]	"abc"[::-1]	Reverses string	'cba'	Check if a string is a palindrome
split()	"a,b,c".split(',')	Tokenize string	['a', 'b', 'c']	Parse CSV data
join()	'-'.join(['a','b'])	Join list into string	'a-b'	Rebuild string from characters
find()	"abcde".find("cd")	Index of substring	2	Pattern search in string
replace()	"abc".replace("b","x")	Replace substring	'axc'	Text masking or sanitization
isalnum()	"abc123".isalnum()	Alphanumeric check	True	Validate password/input
ord() / chr()	ord('a'), chr(97)	Char-ASCII conversion	(97, 'a')	Caesar cipher / encoding
Counter(str)	Counter("aab")	Frequency map	{'a':2,'b':1}	Check if two strings are anagrams

---

📚 LIST / ARRAY FUNCTIONS

Function	Example	Use	Output	Problem
sum()	sum([1,2,3])	Total of elements	6	Subarray sum
max() / min()	max([1,2,3])	Max value	3	Max profit in stock
sorted()	sorted([3,1,2])	Sort array	[1,2,3]	Required for binary search
reversed()	list(reversed([1,2,3]))	Reverse list	[3,2,1]	Palindrome, backtracking

Function	Example	Use	Output	Problem
enumerate()	list(enumerate(['a','b']))	Index tracking	[(0,'a'),(1,'b')]	Custom index logic
zip()	list(zip([1,2],[3,4]))	Pair lists	[(1,3),(2,4)]	Merge coordinates
any() / all()	any([False, True])	Logical checks	(True, True)	Boolean flag checker
permutations()	list(permutations([1,2]))	All orderings	[(1,2),(2,1)]	Permutation problems
combinations()	list(combinations([1,2,3],2))	All pairs	[(1,2),(1,3),(2,3)]	Combination sum, pairs

---

🔢 MATH FUNCTIONS

Function	Example	Use	Output	Problem
abs()	abs(-4)	Absolute value	4	Distance between points
pow()	pow(2,3)	Exponent	8	Modular exponentiation
divmod()	divmod(7,3)	Quotient & remainder	(2,1)	Time formatting
gcd()	math.gcd(8,12)	GCD	4	Simplify fractions
sqrt()	math.sqrt(16)	Square root	4.0	Euclidean distance
log()	math.log(8,2)	Base-2 log	3.0	Binary tree height

🧮 DICTIONARIES

Function	Example	Use	Output	Problem
defaultdict(list)	d['a'].append(1)	Auto-fill keys	{'a': [1]}	Group anagrams, categorize indices
Counter()	Counter([1,2,2,3])	Frequency count	{2:2,1:1,3:1}	Find majority / duplicates

---

🧻 SETS

Function	Example	Use	Output	Problem
set()	set([1,2,2])	Unique values	{1,2}	Remove duplicates
&	{1,2} & {2,3}	Intersection	{2}	Common elements

Function	Example	Use	Output	Problem
`	`	`{1,2}	{2,3}`	Union
-	{1,2} - {2}	Difference	{1}	Exclude blocked items

⛏️ HEAP (Min Priority Queue)

Function	Example	Use	Output	Problem
heapq	heappush()/heappop()	Min-heap	{heap: [1,3,2], popped: 0}	Top K / Kth smallest
nlargest()	heapq.nlargest(2, [5,1,3])	Top N values	[5,3]	K most frequent elements
nsmallest()	heapq.nsmallest(2, [5,1,3])	Lowest N values	[1,3]	Closest K points

---

📥 DEQUE (Double-Ended Queue)

Function	Example	Use	Output	Problem
deque()	append/popleft()	Fast queue	[2]	BFS, sliding window max

---

🔤 STRING FUNCTION PROBLEM EXAMPLES

---

1. s[::-1] → Reverse a String

Problem:

- *Check if a string is a palindrome.*

Example:

```
def isPalindrome(s: str) -> bool:
    return s == s[::-1]
```

📌 **LeetCode 125: Valid Palindrome**

Clean string and check if it reads the same backward.

---

2. s.split(',') → Split a String

Problem:

- *Parse CSV-formatted input into a list of items.*

Example:

```
def parseCSV(s: str) -> list:
    return s.split(',')
```

📌 **Custom parsing problem** – use case in string tokenization.

---

3. '-'.join([...]) → Join List to String

Problem:

- *Convert a list of strings into a hyphenated string.*

Example:

```
def joinWithHyphen(words: list) -> str:
    return '-'.join(words)
```

📌 Useful in problems where you reconstruct results as a single string.

---

4. s.find(sub) → Find Substring

Problem:

- *Return the first index of a substring or -1 if not found.*

Example:

```
def strStr(haystack: str, needle: str) -> int:
    return haystack.find(needle)
```

📌 **LeetCode 28: Find the Index of the First Occurrence in a String**

---

5. s.replace(old, new) → Replace Substring

Problem:

- *Replace all spaces in a string with %20.*

Example:

```
def urlify(s: str) -> str:
    return s.replace(' ', '%20')
```

📌 Common in interview tasks like URL encoding.

---

6. s.isalnum() → Alphanumeric Check

Problem:

- *Validate that a string contains only alphanumeric characters.*

Example:

```
def isValidPassword(s: str) -> bool:
```

```
    return s.isalnum()
```

📌 **LeetCode 125 variant** – Clean string for palindrome check.

---

7. ord(c) / chr(n) → ASCII Conversions

Problem:

- *Implement a Caesar Cipher by shifting each letter by k.*

Example:

```
def caesarCipher(s: str, k: int) -> str:
    result = []
    for c in s:
        if c.isalpha():
            base = ord('a') if c.islower() else ord('A')
            result.append(chr((ord(c) - base + k) % 26 + base))
        else:
            result.append(c)
    return ''.join(result)
```

📌 Classic cipher logic for shifting characters.

---

8. Counter() from collections → Count Characters

Problem:

- *Check if two strings are anagrams of each other.*

Example:

```
from collections import Counter
def isAnagram(s: str, t: str) -> bool:
    return Counter(s) == Counter(t)
```

📌 **LeetCode 242: Valid Anagram**

📚 **LIST / ARRAY FUNCTION PROBLEM EXAMPLES**

---

1. sum() → Total of Elements

Problem:

- *Find subarrays with a given sum (brute-force).*

```
def hasSubarraySum(nums, target):
    for i in range(len(nums)):
        for j in range(i+1, len(nums)+1):
            if sum(nums[i:j]) == target:
                return True
    return False
```

📌 **LeetCode 560 (brute-force version): Subarray Sum Equals K**

---

2. max() / min() → Find Extremes

Problem:

- *Return the max value in an array.*

```
def maxValue(nums):
    return max(nums)
```

📌 **LeetCode 53 (Kadane's):** Use max() to track the max subarray sum.

---

3. sorted() → Sort an Array

Problem:

- *Return the k weakest rows in a matrix (by number of soldiers).*

```
def kWeakestRows(mat, k):
    strength = [(sum(row), i) for i, row in enumerate(mat)]
    return [i for _, i in sorted(strength)[:k]]
```

📌 **LeetCode 1337: The K Weakest Rows in a Matrix**

---

4. reversed() → Reverse a List

Problem:

- *Reverse an array in Python.*

```
def reverseArray(nums):
    return list(reversed(nums))
```

📌 Used in **array reversal, backtracking, or reversing stack simulation.**

---

5. enumerate() → Index and Value

Problem:

- *Find index of first negative number.*

```
def firstNegative(nums):
    for i, val in enumerate(nums):
        if val < 0:
            return i
    return -1
```

📌 Often used in problems where index matters in loops.

---

6. zip() → Pair Two Lists

Problem:

- *Add two arrays element-wise.*

```
def elementWiseSum(a, b):
    return [x + y for x, y in zip(a, b)]
```

📌 Useful in matrix addition or combining arrays with conditions.

---

7. any() / all() → Logical Conditions

Problem:

- *Check if all numbers in list are positive.*

```
def allPositive(nums):
    return all(n > 0 for n in nums)
```

📌 Good for constraint checks in validation tasks.

---

8. permutations() → All Orderings

Problem:

- *Return all permutations of a list.*

```
from itertools import permutations
def allPerms(nums):
    return list(permutations(nums))
```

📌 **LeetCode 46: Permutations**

---

9. combinations() → All Combinations

Problem:

- *Return all 2-element combinations of an array.*

```
from itertools import combinations

def allPairs(nums):
    return list(combinations(nums, 2))
```

📌 Useful in brute-force 2-sum, triangle counting, or pairing logic.

🔁 LOOPS & FUNCTIONAL TOOLS — Real-World Problem Examples

---

1. map() → Apply Function to All Elements

Problem:

- *Convert a list of integers to strings.*

```
def convertToStrings(nums):
    return list(map(str, nums))
```

📌 **Use Case:** Formatting or pre-processing input for display or sorting.

---

2. filter() → Filter Elements by Condition

Problem:

- *Remove all even numbers from a list.*

```
def removeEvens(nums):
    return list(filter(lambda x: x % 2 != 0, nums))
```

📌 **Use Case:** Keeping elements that meet a certain condition (like filtering valid/invalid values).

---

3. lambda → Anonymous Function

Problem:

- *Sort a list of tuples by second element.*

```
def sortBySecondElement(pairs):
    return sorted(pairs, key=lambda x: x[1])
```

📌 **LeetCode-like Use Case:** Custom sorting in problems like **task scheduling**, **meeting rooms**, or **interval merging**.

---

4. map() + lambda

Problem:

- *Square every number in the list.*

```
def squareAll(nums):
    return list(map(lambda x: x**2, nums))
```

📌 Simplifies transformations in one line — great for quick data manipulation.

---

5. filter() + lambda

Problem:

- *Keep only strings with length > 3.*

```
def filterLongWords(words):
    return list(filter(lambda word: len(word) > 3, words))
```

📌 Used in string validation, data cleanup, or form filtering.

---

6. zip() + map()

Problem:

- *Element-wise sum of two lists.*

```
def addLists(a, b):
    return list(map(lambda x: x[0] + x[1], zip(a, b)))
```

📌 Compact alternative to a loop — useful in matrix row operations or adding coordinates.

---

7. enumerate() + lambda

Problem:

- *Find the first even number and its index.*

```
def firstEven(nums):
    return next((i for i, x in enumerate(nums) if x % 2 == 0), -1)
```

📌 Powerful when used with next() and conditionals for fast lookups.

---

8. all() / any() + map()

Problem:

- *Check if all numbers in a list are positive.*

```
def allPositive(nums):
    return all(map(lambda x: x > 0, nums))
```

📌 Logical testing across lists, often used in validation scenarios.

🔢 MATH / NUMBERS — Function-Based Problem Examples

---

1. abs() → Absolute Value

Problem:

- *Calculate Manhattan Distance between two points.*

```
def manhattan_distance(p1, p2):
    return abs(p1[0] - p2[0]) + abs(p1[1] - p2[1])
```

📌 **Used in:** Grid traversal, pathfinding (e.g., LeetCode 1162 – As Far From Land as Possible)

---

2. pow() → Power Calculation

Problem:

- *Check if a number is a power of 3.*

```
def isPowerOfThree(n):
    i = 0
    while pow(3, i) <= n:
        if pow(3, i) == n:
            return True
        i += 1
    return False
```

📌 **LeetCode 326: Power of Three**

---

3. divmod() → Quotient and Remainder

Problem:

- *Convert seconds into (minutes, seconds) format.*

```
def time_format(seconds):
    minutes, sec = divmod(seconds, 60)
    return f"{minutes}m {sec}s"
```

📌 **Used in:** Time formatting, scheduling, pagination logic.

---

4. math.gcd() → Greatest Common Divisor

Problem:

- *Simplify a fraction by dividing by GCD.*

```
import math

def simplify_fraction(numerator, denominator):
    g = math.gcd(numerator, denominator)
    return (numerator // g, denominator // g)
```

📌 **Used in:** Ratio comparisons, modular arithmetic, reduced forms.

---

5. math.sqrt() → Square Root

Problem:

- *Return the integer square root of a number.*

```
import math

def integer_sqrt(n):
    return int(math.sqrt(n))
```

📌 **LeetCode 69: Sqrt(x)** (although typically done with binary search, this is useful for quick approximation)

---

6. math.log() → Logarithmic Computation

Problem:

- *Find number of digits in a number (base 10).*

```
import math

def count_digits(n):
    return math.floor(math.log(n, 10)) + 1 if n > 0 else 1
```

📌 **Use Case:** Determine number of digits, binary search depth, or tree height.

---

7. round() → Round Floating Points

Problem:

- *Round a GPA value to 2 decimal places.*

```
def round_gpa(gpa):
```

```
    return round(gpa, 2)
```

📌 **Used in:** Financial, scoring, or graphical output formatting.

---

8. min() and max() → Min/Max Values

Problem:

- *Find minimum and maximum values in a list.*

```
def min_max(nums):
    return min(nums), max(nums)
```

📌 Used in: Finding boundaries, sliding window extremes, sorting range.

🧠 **DICTIONARIES / HASHMAPS — Problem Examples by Function**

---

1. dict.get(key, default) → Safe Lookup

Problem:

- *Count character frequency in a string without KeyError.*

```
def charFrequency(s):
    freq = {}
    for c in s:
        freq[c] = freq.get(c, 0) + 1
    return freq
```

📌 **LeetCode Use:** Any frequency-based problem (e.g., Valid Anagram, Group Anagrams)

---

2. key in dict → Membership Check

Problem:

- *Find first repeating element in an array.*

```
def firstDuplicate(nums):
    seen = set()
    for num in nums:
        if num in seen:
            return num
        seen.add(num)
    return -1
```

📌 **LeetCode 961: N-Repeated Element in Size 2N Array**

---

3. .items() → Iterate Key-Value Pairs

Problem:

- *Invert a dictionary: swap keys and values.*

```
def invertDict(d):
    return {v: k for k, v in d.items()}
```

📌 Useful in problems requiring value-to-key lookup.

---

4. defaultdict(list) → Auto-initialize Lists

Problem:

- *Group words by their sorted characters (anagram groups).*

```
from collections import defaultdict
def groupAnagrams(strs):
    anagrams = defaultdict(list)
    for word in strs:
        key = ''.join(sorted(word))
        anagrams[key].append(word)
    return list(anagrams.values())
```

📌 **LeetCode 49: Group Anagrams**

---

5. Counter() → Frequency Count

Problem:

- *Check if two strings are anagrams.*

```
from collections import Counter
def isAnagram(s, t):
    return Counter(s) == Counter(t)
```

📌 **LeetCode 242: Valid Anagram**

---

6. dict.pop(key) → Remove Key and Get Value

Problem:

- *Implement LRU Cache (evict least recently used).*

```
def removeKey(d, key):
    if key in d:
        return d.pop(key)
```

📌 **Used in:** Cache eviction, or key cleanup logic.

---

7. dict.keys() / dict.values() → Extract View

Problem:

- *Find if two dictionaries share any value.*

```
def hasCommonValues(d1, d2):
    return bool(set(d1.values()) & set(d2.values()))
```

📌 Used in matching or intersecting mappings.

---

8. dict.update() → Merge Dictionaries

Problem:

- *Combine multiple dictionaries into one.*

```
def mergeDicts(d1, d2):
    d1.update(d2)
    return d1
```

📌 Use case: Combine scores, configs, etc.

---

9. {} Dictionary Comprehension

Problem:

- *Map each character to its frequency in one line.*

```
def frequencyMap(s):
    return {c: s.count(c) for c in set(s)}
```

📌 One-liner for clean, readable dictionary logic.

🔢 **SETS — Real-World Problem Examples by Function**

---

1. set() → Convert List to Set (Remove Duplicates)

Problem:

- *Remove duplicates from a list of integers.*

```
def removeDuplicates(nums):
    return list(set(nums))
```

📌 **LeetCode 217: Contains Duplicate (brute-force check)**

---

2. in (Membership Test)

Problem:

- *Return the intersection of two lists.*

```
def findCommon(nums1, nums2):
    set1 = set(nums1)
    return [x for x in nums2 if x in set1]
```

📌 **LeetCode 349: Intersection of Two Arrays**

---

3. set.add() → Add Element to Set

Problem:

- *Find first repeating element in an array.*

```
def firstDuplicate(nums):
    seen = set()
    for num in nums:
        if num in seen:
            return num
        seen.add(num)
    return -1
```

📌 Common pattern for cycle detection, duplicates, or visited elements.

---

4. set.remove() or discard()

Problem:

- *Keep removing elements until only one remains (simulation).*

```
def lastRemaining(elements):
    s = set(elements)
    while len(s) > 1:
        s.remove(min(s))
    return s.pop()
```

📌 Simulation-style problems, or for modifying working set.

---

5. set.union(set2) or |

Problem:

- *Combine unique elements from two arrays.*

```
def union(nums1, nums2):
    return list(set(nums1) | set(nums2))
```

📌 **LeetCode 349 variant:** Combine tags or merge nodes.

---

6. set.intersection(set2) or &

Problem:

- *Find common favorite movies between two users.*

```
def commonMovies(user1, user2):
    return list(set(user1) & set(user2))
```

📌 Useful for recommendations, user comparisons.

---

7. set.difference(set2) or -

Problem:

- *Find students who haven't submitted homework.*

```
def pendingStudents(all_students, submitted_students):
    return list(set(all_students) - set(submitted_students))
```

📌 **LeetCode 2215: Find the Difference of Two Arrays**

---

8. set.symmetric_difference(set2) or ^

Problem:

- *Find elements present in only one list (not both).*

```
def uniqueInOneList(nums1, nums2):
    return list(set(nums1) ^ set(nums2))
```

📌 Use case: XOR-like behavior, uncommon values.

---

9. len(set(...)) → Count Unique Elements

Problem:

- *Check if a sentence is a pangram (contains all 26 letters).*

```
def isPangram(sentence):
    return len(set(sentence.lower()) & set("abcdefghijklmnopqrstuvwxyz")) == 26
```

📌 **LeetCode 1832: Check if the Sentence Is Pangram**

---

10. set.issubset()

Problem:

- *Check if all required permissions are in user's permission set.*

```
def hasAllPermissions(required, user):
    return set(required).issubset(set(user))
```

📌 Useful for access control and subset checks.

⛏️ HEAP / PRIORITY QUEUE — Python Problem Examples

Python's heapq module implements a **min-heap** by default.

---

1. heapq.heappush() → Push into Heap

Problem:

- *Maintain a running stream of the **K largest elements** seen so far.*

```
import heapq

def kLargestStream(nums, k):
    heap = []
    for num in nums:
        heapq.heappush(heap, num)
        if len(heap) > k:
            heapq.heappop(heap)
    return heap
```

📌 **LeetCode 703: Kth Largest Element in a Stream**

---

2. heapq.heappop() → Pop Min Element

Problem:

- *Find the Kth smallest number in a list.*

```
def kthSmallest(nums, k):
    heapq.heapify(nums)
    for _ in range(k - 1):
        heapq.heappop(nums)
    return heapq.heappop(nums)
```

📌 **LeetCode 215 variant:** Get smallest elements by popping.

---

3. heapq.heapify() → Convert List to Min-Heap

Problem:

- *Sort a nearly-sorted array (elements are k positions away).*

```
def sortKSortedArray(arr, k):
    heap = arr[:k+1]
    heapq.heapify(heap)
    result = []
    for i in range(k+1, len(arr)):
        result.append(heapq.heappushpop(heap, arr[i]))
    while heap:
        result.append(heapq.heappop(heap))
    return result
```

📌 **LeetCode 253 or 621 variants:** Efficiently manage task scheduling or merging.

---

4. heapq.heappushpop() → Push then Pop (Faster than separate push & pop)

Problem:

- *Keep smallest k numbers using optimized heap operations.*

```
def kSmallestOptimized(nums, k):
    heap = []
    for num in nums:
        if len(heap) < k:
            heapq.heappush(heap, -num)
        else:
```

```
        heapq.heappushpop(heap, -num)
    return [-x for x in heap]
```

📌 Optimized for performance-sensitive problems.

---

5. heapq.nlargest(k, iterable) → Get Top K Largest Elements

Problem:

- *Find top K frequent elements in an array.*

```
from collections import Counter

def topKFrequent(nums, k):
    count = Counter(nums)
    return heapq.nlargest(k, count.keys(), key=count.get)
```

📌 **LeetCode 347: Top K Frequent Elements**

---

6. heapq.nsmallest(k, iterable) → Get K Smallest Elements

Problem:

- *Find the K closest points to origin in 2D plane.*

```
def kClosest(points, k):
    return heapq.nsmallest(k, points, key=lambda p: p[0]**2 + p[1]**2)
```

📌 **LeetCode 973: K Closest Points to Origin**

---

7. Max-Heap via Negation Trick

Problem:

- *Simulate max-heap using min-heap + negation (common in interviews).*

```
def maxHeapExample(nums):
    max_heap = [-n for n in nums]
    heapq.heapify(max_heap)
    return -heapq.heappop(max_heap)
```

📌 Needed in many problems (e.g. **find Kth largest**, **median sliding window**) since heapq is only a min-heap.

📩 collections.deque — Real-World Problem Examples by Function

deque is optimized for **O(1)** insertions/removals from both ends.

---

1. deque.append() → Add to Right (Tail)

Problem:

- 🔹 *Implement a basic queue (FIFO).*

```
from collections import deque


def basicQueue(ops):
    q = deque()
    for op in ops:
        if op[0] == 'push':
            q.append(op[1])
        elif op[0] == 'pop':
            if q:
                q.popleft()
    return list(q)
```

📌 Used in **queue simulations**, scheduling, stream processing.

---

2. deque.appendleft() → Add to Left (Head)

Problem:

- 🔹 *Simulate a browser back-stack with instant "push to front".*

```
def simulateBackStack(pages):
    dq = deque()
    for page in pages:
        dq.appendleft(page)  # newest page at front
    return list(dq)
```

📌 Use case: **LRU Cache**, recent items, BFS level reversal.

---

3. deque.pop() → Remove from Right (Tail)

Problem:

- 🔹 *Remove elements from a deque to maintain a decreasing order (monotonic stack logic).*

```
def removeSmallerRight(nums):
    dq = deque()
    for num in nums:
        while dq and dq[-1] < num:
            dq.pop()
        dq.append(num)
    return list(dq)
```

📌 Used in **sliding window maximum** (LeetCode 239).

---

4. deque.popleft() → Remove from Left (Head)

Problem:

🔹 *Perform level order traversal (BFS) of a binary tree.*

```
from collections import deque


def levelOrder(root):
    if not root:
        return []
    dq = deque([root])
    result = []
    while dq:
        level = []
        for _ in range(len(dq)):
            node = dq.popleft()
            level.append(node.val)
            if node.left:
                dq.append(node.left)
            if node.right:
                dq.append(node.right)
        result.append(level)
    return result
```

📌 **LeetCode 102: Binary Tree Level Order Traversal**

---

5. deque for Sliding Window Problems

Problem:

- *Find maximum in each sliding window of size k.*

```
def maxSlidingWindow(nums, k):
    dq = deque()
    result = []
    for i in range(len(nums)):
        while dq and dq[0] <= i - k:
            dq.popleft()
        while dq and nums[dq[-1]] < nums[i]:
            dq.pop()
        dq.append(i)
        if i >= k - 1:
            result.append(nums[dq[0]])
    return result
```

📌 **LeetCode 239: Sliding Window Maximum** — classic deque use!

---

6. deque.rotate(k) → Rotate Elements (Right by k)

Problem:

- *Rotate an array by k steps to the right.*

```
def rotateArray(nums, k):
    dq = deque(nums)
    dq.rotate(k)
    return list(dq)
```

📌 **LeetCode 189: Rotate Array**

---

7. deque.clear() → Remove All Elements

Problem:

- *Reset the queue when a condition is met (e.g., overflow).*

```
def resetOnThreshold(ops, threshold):
    dq = deque()
    for op in ops:
        dq.append(op)
        if len(dq) > threshold:
            dq.clear()
    return list(dq)
```

📌 Used in bounded buffer, event streams.

Summary of Deque Functions in Action

Function	Common Use Case
append()	Queue operations, level order
appendleft()	LRU cache, reversing order
pop()	Maintain decreasing/increasing stack
popleft()	BFS traversal, sliding window
rotate(k)	Array rotation
clear()	Reset buffers or queues

🌐 **GRAPH TRAVERSAL (BFS / DFS)**

---

1. BFS (Breadth-First Search) using deque

🔹 **Problem:** *Find the shortest path in a binary matrix.*

```
from collections import deque

def shortestPathBinaryMatrix(grid):
    n = len(grid)
    if grid[0][0] or grid[-1][-1]: return -1
    dq = deque([(0, 0, 1)])
    visited = set((0, 0))
```

```
    while dq:
        x, y, dist = dq.popleft()
        if x == y == n - 1:
            return dist
        for dx, dy in [(-1, -1), (-1, 0), ..., (1, 1)]:
            nx, ny = x + dx, y + dy
            if 0 <= nx < n and 0 <= ny < n and grid[nx][ny] == 0 and (nx, ny) not in visited:
                dq.append((nx, ny, dist + 1))
                visited.add((nx, ny))
    return -1
```

📌 **LeetCode 1091**

---

2. DFS (Depth-First Search) using Recursion

🔹 **Problem:** *Number of islands in a grid (count connected components).*

```
def numIslands(grid):
    def dfs(r, c):
        if 0 <= r < len(grid) and 0 <= c < len(grid[0]) and grid[r][c] == '1':
            grid[r][c] = '0'
            for dr, dc in [(-1,0), (1,0), (0,-1), (0,1)]:
                dfs(r + dr, c + dc)

    count = 0
    for r in range(len(grid)):
        for c in range(len(grid[0])):
            if grid[r][c] == '1':
                dfs(r, c)
                count += 1
    return count
```

📌 **LeetCode 200: Number of Islands**

---

3. Topological Sort (Kahn's Algorithm - BFS with Indegree)

🔹 **Problem:** *Course schedule - can all courses be finished?*

```
from collections import deque, defaultdict

def canFinish(numCourses, prerequisites):
    indegree = [0] * numCourses
    graph = defaultdict(list)

    for dest, src in prerequisites:
        graph[src].append(dest)
        indegree[dest] += 1

    dq = deque([i for i in range(numCourses) if indegree[i] == 0])

    while dq:
        node = dq.popleft()
        numCourses -= 1
        for neighbor in graph[node]:
            indegree[neighbor] -= 1
            if indegree[neighbor] == 0:
                dq.append(neighbor)

    return numCourses == 0
```

📌 **LeetCode 207: Course Schedule**

🔁 **BACKTRACKING PATTERNS**

---

1. Subsets (Power Set)

🔹 **Problem:** *Return all subsets of a list.*

```
def subsets(nums):
    res = []
    def backtrack(start, path):
```

```
        res.append(path[:])
        for i in range(start, len(nums)):
            path.append(nums[i])
            backtrack(i + 1, path)
            path.pop()
    backtrack(0, [])
    return res
```

📌 **LeetCode 78: Subsets**

---

2. Permutation Generation

🔹 **Problem:** *Return all permutations of an array.*

```
def permute(nums):
    res = []
    def backtrack(path, used):
        if len(path) == len(nums):
            res.append(path[:])
            return
        for i in range(len(nums)):
            if used[i]: continue
            used[i] = True
            path.append(nums[i])
            backtrack(path, used)
            path.pop()
            used[i] = False
    backtrack([], [False] * len(nums))
    return res
```

📌 **LeetCode 46: Permutations**

---

3. N-Queens Solver

🔹 **Problem:** *Place N queens on an N×N chessboard such that none attack each other.*

```
def solveNQueens(n):
```

```
    res = []
    def backtrack(r, cols, diag1, diag2, board):
        if r == n:
            res.append([''.join(row) for row in board])
            return
        for c in range(n):
            if c in cols or r + c in diag1 or r - c in diag2:
                continue
            board[r][c] = 'Q'
            backtrack(r + 1, cols | {c}, diag1 | {r + c}, diag2 | {r - c}, board)
            board[r][c] = '.'
    board = [['.'] * n for _ in range(n)]
    backtrack(0, set(), set(), set(), board)
    return res
```

📌 **LeetCode 51: N-Queens**

💡 **DYNAMIC PROGRAMMING PATTERNS**

---

1. 0/1 Knapsack (Classic DP Table)

🔹 **Problem:** *Find max value that fits into weight capacity.*

```
def knapsack(weights, values, capacity):
    n = len(weights)
    dp = [[0]*(capacity + 1) for _ in range(n+1)]
    for i in range(1, n+1):
        for w in range(capacity + 1):
            if weights[i-1] <= w:
                dp[i][w] = max(dp[i-1][w], values[i-1] + dp[i-1][w - weights[i-1]])
            else:
                dp[i][w] = dp[i-1][w]
    return dp[n][capacity]
```

---

2. Longest Increasing Subsequence (LIS)

🔹 **Problem:** *Find length of longest increasing subsequence.*

```
def lengthOfLIS(nums):
    dp = [1]*len(nums)
    for i in range(len(nums)):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] = max(dp[i], dp[j]+1)
    return max(dp)
```

📌 **LeetCode 300: Longest Increasing Subsequence**

---

3. Memoized Fibonacci (Top-Down DP)

🔹 **Problem:** *Return the nth Fibonacci number with memoization.*

```
from functools import lru_cache

@lru_cache(None)
def fib(n):
    if n <= 1: return n
    return fib(n - 1) + fib(n - 2)
```

📌 **LeetCode 509: Fibonacci Number**

---

4. Tabulated Fibonacci (Bottom-Up DP)

```
def fib(n):
    if n <= 1: return n
    dp = [0, 1]
    for i in range(2, n+1):
        dp.append(dp[-1] + dp[-2])
    return dp[n]
```

---

5. DP on Strings: Longest Common Subsequence (LCS)

```
def longestCommonSubsequence(text1, text2):
```

```
    m, n = len(text1), len(text2)
    dp = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m):
        for j in range(n):
            if text1[i] == text2[j]:
                dp[i+1][j+1] = dp[i][j]+1
            else:
                dp[i+1][j+1] = max(dp[i][j+1], dp[i+1][j])
    return dp[m][n]
```

📌 **LeetCode 1143: LCS**

🧠 **Top Useful Algorithms + Example Problems**

---

1. Binary Search

🔹 **When to Use:** Sorted arrays, search space reduction, optimization problems.
🔍 **Problem:** *Find square root of a number (integer part only)*

```
def sqrt(x):
    left, right = 0, x
    while left <= right:
        mid = (left + right) // 2
        if mid * mid <= x < (mid + 1) * (mid + 1):
            return mid
        elif mid * mid < x:
            left = mid + 1
        else:
            right = mid - 1
```

📌 **LeetCode 69: Sqrt(x)**

---

2. Two Pointers

🔹 **When to Use:** Sorted arrays, linked lists, palindrome checks, in-place operations.
🔍 **Problem:** *Check if a string is a palindrome ignoring non-alphanumeric.*

```
def isPalindrome(s):
    s = [c.lower() for c in s if c.isalnum()]
    left, right = 0, len(s) - 1
    while left < right:
        if s[left] != s[right]:
            return False
        left += 1
        right -= 1
    return True
```

📌 **LeetCode 125: Valid Palindrome**

---

3. Sliding Window

🔹 **When to Use:** Substrings, subarrays, continuous window problems.
🔍 **Problem:** *Find longest substring without repeating characters*

```
def lengthOfLongestSubstring(s):
    seen = {}
    left = 0
    max_len = 0
    for right in range(len(s)):
        if s[right] in seen and seen[s[right]] >= left:
            left = seen[s[right]] + 1
        seen[s[right]] = right
        max_len = max(max_len, right - left + 1)
    return max_len
```

📌 **LeetCode 3: Longest Substring Without Repeating Characters**

---

4. Prefix Sum

🔹 **When to Use:** Fast range sum queries, subarrays with target sum.
🔍 **Problem:** *Find number of subarrays that sum to k*

```
from collections import defaultdict
```

```
def subarraySum(nums, k):
    prefix = defaultdict(int)
    prefix[0] = 1
    curr_sum = 0
    count = 0
    for num in nums:
        curr_sum += num
        count += prefix[curr_sum - k]
        prefix[curr_sum] += 1
    return count
```

📌 **LeetCode 560: Subarray Sum Equals K**

---

5. Greedy Algorithm

🔹 **When to Use:** Local optimal choices lead to global optimum.
🔍 **Problem:** *Find minimum number of arrows to burst balloons.*

```
def findMinArrowShots(points):
    points.sort(key=lambda x: x[1])
    arrows = 1
    end = points[0][1]
    for start, stop in points[1:]:
        if start > end:
            arrows += 1
            end = stop
    return arrows
```

📌 **LeetCode 452: Minimum Number of Arrows to Burst Balloons**

---

6. Union-Find (Disjoint Set)

🔹 **When to Use:** Detecting cycles, connected components in graphs.
🔍 **Problem:** *Detect cycle in an undirected graph*

```
def find(parent, x):
    if parent[x] != x:
```

```
        parent[x] = find(parent, parent[x])
    return parent[x]

def union(parent, rank, x, y):
    rootX = find(parent, x)
    rootY = find(parent, y)
    if rootX == rootY: return False
    if rank[rootX] < rank[rootY]:
        parent[rootX] = rootY
    else:
        parent[rootY] = rootX
        if rank[rootX] == rank[rootY]:
            rank[rootX] += 1
    return True

def hasCycle(edges, n):
    parent = [i for i in range(n)]
    rank = [0] * n
    for u, v in edges:
        if not union(parent, rank, u, v):
            return True
    return False
```

📌 **Used in:** Kruskal's algorithm, connectivity problems.

---

7. Backtracking

🔹 **When to Use:** Try-all-combinations with pruning.
🔍 **Problem:** *Generate all combinations of well-formed parentheses*

```
def generateParenthesis(n):
    res = []
    def backtrack(s, left, right):
        if len(s) == 2 * n:
```

```
            res.append(s)
            return
        if left < n:
            backtrack(s + '(', left + 1, right)
        if right < left:
            backtrack(s + ')', left, right + 1)
    backtrack('', 0, 0)
    return res
```

📌 **LeetCode 22: Generate Parentheses**

---

8. Dynamic Programming

🔹 **When to Use:** Overlapping subproblems, optimal substructure.
🔍 **Problem:** *Climbing stairs – count ways to climb n steps*

```
def climbStairs(n):
    if n <= 2: return n
    a, b = 1, 2
    for _ in range(3, n+1):
        a, b = b, a + b
    return b
```

📌 **LeetCode 70: Climbing Stairs**

---

9. Topological Sort (Graph)

🔹 **When to Use:** Task scheduling, course prerequisites.
🔍 **Problem:** *Return a valid order to finish all courses.*
(Example already shown earlier using BFS)

📌 **LeetCode 210: Course Schedule II**

---

10. Kadane's Algorithm

🔹 **When to Use:** Maximum subarray sum (single scan).
🔍 **Problem:** *Find the contiguous subarray with largest sum*

```
def maxSubArray(nums):
    curr_sum = max_sum = nums[0]
```

```
    for num in nums[1:]:
        curr_sum = max(num, curr_sum + num)
        max_sum = max(max_sum, curr_sum)
    return max_sum
```

📌 **LeetCode 53: Maximum Subarray**
