DSA Interview Notes – Standard Layouts & Mappings

1. Phone Keypad Mapping

```
phone_map = {
    "2": "abc", "3": "def",
    "4": "ghi", "5": "jkl",
    "6": "mno", "7": "pqrs",
    "8": "tuv", "9": "wxyz"
}
```

✓ Use in:

- Letter Combinations of Phone Number, Predictive Text / T9, Find All Possible Words from Digits
- Backtracking, DFS, Strings

---

2. Grid/Matrix Directions

4-Direction Movement (Up, Down, Left, Right)

```
dirs_4 = [(-1, 0), (1, 0), (0, -1), (0, 1)]
```

8-Direction Movement (Diagonals included)

```
dirs_8 = [(-1, -1), (-1, 0), (-1, 1),
          (0, -1),           (0, 1),
          (1, -1),  (1, 0),  (1, 1)]
```

✓ Use in:

- Number of Islands, Word Search, Maze Solving, Flood Fill
- DFS/BFS, Matrix Problems

---

3. Knight’s Moves (Chessboard)

```
knight_moves = [
    (2, 1), (1, 2), (-1, 2), (-2, 1),
    (-2, -1), (-1, -2), (1, -2), (2, -1)
]
```

✓ Use in: Knight’s Tour, Minimum Knight Moves , Puzzle/Chess Grid

---

4. Binary Tree Traversals

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

- **Preorder**: Root → Left → Right
- **Inorder**: Left → Root → Right
- **Postorder**: Left → Right → Root
- **Level Order**: BFS using queue

✅ Use in: Tree reconstruction, Path Sum, BST validation, Trees, Recursion, DFS

---

5. Min Heap / Max Heap in Python

```
import heapq

# Min Heap
heapq.heappush(heap, val)
heapq.heappop(heap)

# Max Heap
heapq.heappush(heap, -val)
heapq.heappop(heap) * -1
```

✅ Use in: Top K elements, Median in stream, Kth smallest/largest, Greedy, Priority Queues

---

6. ASCII Mapping (Characters → Numbers)

```
ord('a')   # 97
chr(97)    # 'a'

ord('z')   # 122
```

✅ Use in: Frequency Arrays (size 26), Anagrams, Palindrome Check , Frequency counting

---

7. Keyboard Row Mapping

```
row1 = set("qwertyuiop")
row2 = set("asdfghjkl")
row3 = set("zxcvbnm")
```

✅ Use in: Words Using Only One Row of Keyboard, String filtering

---

8. Roman Numerals Mapping

```
roman_map = {
    'I': 1, 'V': 5, 'X': 10,
    'L': 50, 'C': 100, 'D': 500, 'M': 1000
}
```

✓ Use in: Roman to Integer, Integer to Roman, String / Math Conversion

---

9. Weekday Layout (for calendar problems)

```
weekdays = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday']
```

✓ Use in: Day calculation, Zeller's Congruence

---

10. Digit to Word Mapping

```
digit_to_word = {
  0: 'zero', 1: 'one', ..., 9: 'nine'
}
```

✓ Use in: Verbal arithmetic puzzles , Spoken digit output problems

---

The Top 14 LeetCode Patterns You Need to Know

1. Sliding Window

✅ Used when working with **subarrays, substrings**, or **fixed-size / variable windows**.

Helps in reducing time from $O(n^2) \rightarrow O(n)$

Examples: Maximum Sum Subarray of Size K , Longest Substring Without Repeating Characters, Minimum Window Substring

---

2. Two Pointers

✅ Used for problems involving **sorted arrays**, **linked lists**, etc.

Moves two pointers from front/back/middle

Examples: Two Sum II (sorted array), 3Sum, Container With Most Water

---

3. Fast and Slow Pointers

✅ Detect **cycles**, mid-points

Examples: Linked List Cycle, Find the Duplicate Number , Middle of Linked List

---

4. Hashing (HashMap / Set)

✅ Count frequencies, track seen elements, prefix sums

Examples: Subarray Sum Equals K , Group Anagrams , Longest Consecutive Sequence

---

5. Prefix Sum

✅ For cumulative/interval sums, range queries

Examples: Subarray Sum Equals K , Range Sum Query , Count Subarrays with Sum

---

6. Backtracking

✅ Try all combinations, revert decisions

Examples: Subsets , Permutations , Letter Combinations of a Phone Number, N-Queens

7. Recursion / DFS

✅ Tree/graph traversal, subset problems

Examples: Binary Tree Traversals , Generate Parentheses , Subsets

8. Breadth-First Search (BFS)

✅ Used in graphs, trees, shortest paths

Examples: Binary Tree Level Order Traversal, Word Ladder, Number of Islands

9. Depth-First Search (DFS)

✅ Graph traversal, backtracking, component counting

Examples: Number of Islands , Clone Graph , Word Search

10. Greedy

✅ Make best local choice at each step

Examples: Jump Game , Merge Intervals , Gas Station , Partition Labels

11. Dynamic Programming (DP)

✅ Break down problem into subproblems, store results

Examples: House Robber , Longest Increasing Subsequence , 0/1 Knapsack, Coin Change

12. Bit Manipulation

✅ Work with bits for optimization or tricky logic

Examples: Single Number , Counting Bits , Subsets (with bitmasking)

13. Union Find (Disjoint Set)

✅ Track connected components

Examples: Graph Valid Tree , Number of Connected Components, Redundant Connection

---

14. Heap / Priority Queue

✅ Get max/min quickly, sliding window max, k-largest

Examples: Kth Largest Element, Merge K Sorted Lists, Top K Frequent Elements

---

Extra Patterns (Advanced – optional):

If you **have time later**, explore these **bonus topics**:

1. Monotonic Stack / Deque
2. Segment Trees / Binary Indexed Trees
3. Dijkstra's / A* for weighted graphs
4. Tries (prefix trees)
5. Rabin-Karp (String Hashing)

These are **not Amazon must-knows**, but helpful for:

- Google
- Deep graph / string / optimization problems

---

1. Built-in Functions (Essentials)

Function	Purpose	Example
max()	Get the maximum of values	max(3, 7) → 7
min()	Get the minimum	min(3, 7) → 3
sum()	Sum of list values	sum([1, 2, 3]) → 6
len()	Length of list, string, etc.	len(arr)
range()	Generate a sequence	range(5) → 0, 1, 2, 3, 4
enumerate()	Loop with index	for i, val in enumerate(arr)
reversed()	Reverse iterator	for val in reversed(arr)
sorted()	Returns sorted version of list	sorted(arr)
list()	Convert to list	list("abc") → ['a','b','c']

---

2. Strings – Helpful Methods

Function	Purpose	Example
str.split()	Split string into list	"a b c".split() → ['a','b','c']
str.strip()	Remove whitespace	" abc ".strip() → "abc"
str.isdigit()	Check if string is a digit	"123".isdigit() → True
str.lower() / upper()	Convert case	"HeLLo".lower() → "hello"

---

3. Lists – Core Methods

Method	Purpose	Example
list.append(x)	Add element at the end	arr.append(10)
list.pop()	Remove last element	arr.pop()
list.remove(x)	Remove element by value	arr.remove(2)
list.index(x)	Get index of value	arr.index(5)
list.insert(i, x)	Insert at index	arr.insert(1, 99)
list.count(x)	Count occurrences	arr.count(2)

4. Dictionary (Hashmap) – Extremely Useful

Method / Function	Purpose	Example
dict.get(key, default)	Get value or return default	d.get('a', 0)
dict.keys() / values()	Iterate keys/values	for key in d.keys()
dict.items()	Iterate key, value pairs	for k, v in d.items()
key in dict	Check if key exists	'a' in d
defaultdict(int)	Auto-handle missing keys	from collections import defaultdict

5. Collections Module

Tool	Purpose	Example
defaultdict(type)	Dictionary with default values	dd = defaultdict(int)
Counter()	Count occurrences of elements	Counter("aabc") → {'a':2, 'b':1,...}
deque()	Double-ended queue	dq = deque([1,2,3])
heapq	Priority Queue / Min Heap	heapq.heappush(heap, val)

6. Math / Time / Random

Module	Function	Purpose
math	math.sqrt(), math.ceil()	Square root, rounding
time	time.time()	Get current time (for speed)
random	random.randint()	Random numbers (for mock tests)

---

7. Useful Tricks

Trick	Description	Example
arr[::-1]	Reverse a list or string	s[::-1]
a, b = b, a	Swap variables	
' '.join(list)	Convert list to string with spaces	' '.join(['a','b']) → "a b"
all() / any()	Check all / any condition is True	all([1, True])

Bonus: Lambda & Key Functions (for Sorting)

```
arr.sort(key=lambda x: x[1])  # Sort by second element of sublists
```

Useful for:

- Sorting arrays of tuples
- Sorting by custom logic

Great question, Bhawana! You're already covering a solid base. But yes — here's a **final roundup of additional useful Python concepts and functions** that are **often overlooked but critical** in real DSA problems (especially in contests and interviews like Amazon).

---

8. Advanced Built-ins & Tricks (Most Students Miss These!)

Feature/Function	Use Case	Example
`zip()`	Loop over multiple iterables simultaneously	`for a, b in zip(arr1, arr2)`
`map()`	Apply function to each item	`map(int, input().split())`
`filter()`	Filter elements by condition	`filter(lambda x: x > 0, arr)`
`set()`	Removes duplicates, used for fast lookup	`set(arr)`
`set.add()` / `set.remove()`	Modify set	
`islice()` from `itertools`	Slicing iterators	`islice(range(100), 10, 20)`
`bisect.bisect_left()` / `bisect_right()`	Binary search in sorted list	`bisect.bisect_left(arr, x)`
`any()`	True if any element is truthy	`any([False, True, False])` → `True`
`all()`	True if all elements are truthy	`all([1, 2, 3])` → `True`
`re` module	Regex for pattern matching (sometimes needed in parsing)	`re.findall(r'\d+', s)`

---

9. Built-in Data Types You Must Master

Type	When to Use	Quick Syntax Example
`list`	Ordered, indexable	`a = [1, 2, 3]`
`tuple`	Immutable, hashable keys	`a = (1, 2)`
`set`	Unique values, fast lookup	`a = {1, 2, 3}`
`dict`	Key-value mapping	`d = {'a': 1}`
`deque`	Queue/Stack with fast ops	`deque([1,2,3])` from `collections`
`heap`	Min/Max heap for priority	`heapq.heappush(heap, val)`

---

10. Collections Deep Dive (Very Interview-Friendly)

Tool	Use Case	Example
`defaultdict(list/int)`	Auto-initialize missing keys	`dd = defaultdict(list)`
`Counter()`	Count occurrences of each element	`Counter([1,2,2,3]) → {2:2, 1:1, 3:1}`
`OrderedDict()`	Keeps insertion order	(Rarely used now, since dicts are ordered from Python 3.7+)
`deque()`	Queue or two-sided operations	`dq.popleft(), dq.appendleft()`

---

11. heapq (Min/Max Heap)

```
import heapq
min_heap = []
heapq.heappush(min_heap, 5)
heapq.heappush(min_heap, 3)
heapq.heappop(min_heap)  # returns 3 (smallest)
```

Use in:
✓ Top-K problems,
✓ Priority queues,
✓ Greedy algorithms

---

12. Functools (for memoization and recursion)

```
from functools import lru_cache

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

Use in:
✓ **Dynamic Programming**,
✓ **Top-down memoization**,
✓ **Recursion-heavy problems**

13. Bit Manipulation Functions

Bit Trick	Purpose
`x << 1`, `x >> 1`	Bit shifts (multiply/divide by 2)
`x & 1`	Check if x is odd
`x ^ y`	Bitwise XOR
`bin(x).count('1')`	Count set bits in x

Python Notes for Beginners to Intermediate

1. Introduction to Python

- **High-level**, **interpreted**, and **dynamically typed** programming language.
- Created by **Guido van Rossum**, released in **1991**.
- Great for web dev, data science, automation, AI, scripting, etc.

---

2. Basic Syntax

```
# This is a comment
print("Hello, World!")  # Output: Hello, World!
```

- **Indentation** is mandatory in Python to define blocks.
- No `{}` or `;` like in C/C++/Java.

---

3. Data Types

- `int`, `float`, `str`, `bool`
- `list`, `tuple`, `set`, `dict`
- `NoneType`

```
a = 5                # int
b = 5.0              # float
c = "Python"         # str
d = True             # bool
```

---

4. Type Conversion

```
int("5")         # 5
float("3.14")    # 3.14
str(100)         # "100"
bool(0)          # False
```

---

5. Variables

- No need to declare the type.
- Follows snake_case naming.

```
name = "Bhawana"
age = 25
```

6. Control Flow

if / elif / else

```
if age > 18:
    print("Adult")
elif age == 18:
    print("Just turned adult")
else:
    print("Minor")
```

while loop

```
i = 0
while i < 5:
    print(i)
    i += 1
```

for loop

```
for i in range(5):
    print(i)
```

7. Functions

```
def greet(name):
    return f"Hello, {name}"

print(greet("Bhawana"))
```

8. Strings

```
text = "Python"
print(text.upper())        # 'PYTHON'
print(text[0])             # 'P'
print(len(text))           # 6
print("th" in text)        # True
```

9. Lists

```
fruits = ["apple", "banana", "mango"]
fruits.append("orange")
print(fruits[1])           # "banana"
```

10. Tuples

```
t = (1, 2, 3)
# Immutable
print(t[0])               # 1
```

---

11. Dictionaries

```
student = {"name": "Alice", "age": 20}
print(student["name"])    # Alice
student["age"] = 21
```

---

12. Sets

```
nums = {1, 2, 3, 2}
print(nums)               # {1, 2, 3} - duplicates removed
```

---

13. Exception Handling

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Can't divide by zero!")
finally:
    print("Done")
```

---

14. Modules and Libraries

```
import math
print(math.sqrt(16))      # 4.0
```

Install external modules:

```
pip install requests
```

---

15. List Comprehension

```
squares = [x**2 for x in range(5)]   # [0, 1, 4, 9, 16]
```

---

16. Lambda Functions

```
square = lambda x: x ** 2
print(square(5))          # 25
```

17. Object-Oriented Programming (OOP)

```
class Person:
    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print(f"Hi, I'm {self.name}")

p = Person("Bhawana")
p.say_hi()
```

18. File Handling

```
with open("file.txt", "r") as file:
    content = file.read()
    print(content)
```

19. Useful Built-in Functions

- `len()`, `type()`, `range()`, `sorted()`, `input()`, `sum()`, `max()`, `min()`

20. Best Practices

- Use meaningful variable names.
- Keep code DRY (Don't Repeat Yourself).
- Use virtual environments.
- Follow PEP8 (Python Style Guide).