Member-only story

Crack FAANG Interviews: 30 Essential Python Questions!

Ritesh Gupta · Follow
Published in Artificial Intelligence in Plain English · 15 min read · Oct 1, 2024

If you're preparing for a coding interview at one of the **FAANG** (Facebook, Amazon, Apple, Netflix, Google) companies, get ready! The competition is fierce, the expectations are high, and Python will likely be your best friend 🐍. Python is fast, flexible, and clean — qualities you'll need for those tricky interview questions.

To help you on this journey, I've compiled **20 of the most commonly asked Python interview questions** that pop up at FAANG interviews. And don't

worry, I'll include code examples and break them down in a way that's fun and human-friendly. Let's get started!

1. Reverse a String 🧿

This question might seem basic, but it's super common! It checks your understanding of Python slicing.

🔍 **Question:** Write a function to reverse a string.

```
def reverse_string(s: str) -> str:
    return s[::-1]

# Example
print(reverse_string("hello"))  # Output: "olleh"
```

Why it's important: This tests your knowledge of string manipulation and Python's neat slicing features.

2. Check for Anagrams 🔀

Two strings are anagrams if they contain the same characters in different orders.

🔍 **Question:** Write a function to check if two strings are anagrams.

```
from collections import Counter

def are_anagrams(str1: str, str2: str) -> bool:
    return Counter(str1) == Counter(str2)

# Example
print(are_anagrams("listen", "silent"))  # Output: True
```

Why it's important: Anagram problems test your understanding of hash maps and data structures like dictionaries.

3. Find the First Non-Repeating Character 😶

It's a classic problem that comes up often. FAANG loves this one because it tests your string traversal and map skills.

🔍 **Question:** Write a function to find the first non-repeating character in a string.

```
def first_non_repeating_char(s: str) -> str:
    freq = {}
    for char in s:
        freq[char] = freq.get(char, 0) + 1

    for char in s:
        if freq[char] == 1:
            return char
    return None

# Example
print(first_non_repeating_char("swiss"))  # Output: "w"
```

Why it's important: This checks your understanding of linear time solutions using dictionaries or hash maps.

4. Merge Two Sorted Lists 🧩

It's a classic merging problem. They're testing your ability to handle lists in sorted order.

🔍 **Question:** Write a function to merge two sorted lists into one sorted list.

```
def merge_sorted_lists(list1: list, list2: list) -> list:
    return sorted(list1 + list2)

# Example
print(merge_sorted_lists([1, 3, 5], [2, 4, 6]))  # Output: [1, 2, 3, 4, 5, 6]
```

Why it's important: This question checks your ability to merge and sort lists efficiently.

5. Implement Fibonacci Recursively 🌀

Ah, recursion. You'll need this for problems involving dynamic programming or mathematical sequences.

🔍 **Question:** Write a recursive function to compute the nth Fibonacci number.

```python
def fibonacci(n: int) -> int:
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

# Example
print(fibonacci(5))  # Output: 5
```

Why it's important: Recursion is a critical skill, and FAANG loves testing whether you can optimize with memoization.

6. Find the Maximum Subarray Sum 💪

This one is known as **Kadane's Algorithm**. It tests your ability to work with arrays and optimize performance.

🔍 **Question**: Write a function to find the maximum sum of a contiguous subarray.

```python
def max_subarray_sum(arr: list) -> int:
    current_sum = max_sum = arr[0]

    for num in arr[1:]:
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)

    return max_sum

# Example
print(max_subarray_sum([-2,1,-3,4,-1,2,1,-5,4]))  # Output: 6
```

Why it's important: This is a great way to test dynamic programming and array traversal.

7. Find Missing Number in an Array 🔢

You'll often face array problems like this, and it tests your mathematical and logic skills.

🔍 **Question**: Given an array of size `n` containing numbers from 1 to `n+1` with one missing, find the missing number.

```python
def find_missing_number(arr: list) -> int:
    n = len(arr) + 1
    total_sum = n * (n + 1) // 2
```

```
    return total_sum - sum(arr)

# Example
print(find_missing_number([1, 2, 4, 5, 6]))  # Output: 3
```

Why it's important: It's a brain teaser but easy to solve with a mathematical formula.

8. Find the Longest Common Prefix 🎯

An interview classic! Finding the longest common prefix among a list of strings is a great test of string manipulation.

🔍 **Question**: Write a function to find the longest common prefix among an array of strings.

```
def longest_common_prefix(strs: list) -> str:
    if not strs:
        return ""

    prefix = strs[0]
    for string in strs[1:]:
        while not string.startswith(prefix):
            prefix = prefix[:-1]
            if not prefix:
                return ""
    return prefix

# Example
print(longest_common_prefix(["flower", "flow", "flight"]))  # Output: "fl"
```

Why it's important: String manipulation, loop control, and the concept of reducing search space — super useful skills.

9. Rotate a Matrix by 90 Degrees 🔄

Working with matrices is a popular theme in FAANG interviews. Rotating a matrix is a neat challenge.

🔍 **Question**: Write a function to rotate a matrix by 90 degrees.

```
def rotate_matrix(matrix: list) -> list:
    return [list(row) for row in zip(*matrix[::-1])]

# Example
matrix = [
    [1, 2, 3],
```

```
    [4, 5, 6],
    [7, 8, 9]
]
rotated_matrix = rotate_matrix(matrix)
print(rotated_matrix)
# Output:
# [[7, 4, 1],
#  [8, 5, 2],
#  [9, 6, 3]]
```

Why it's important: This tests your knowledge of nested loops, matrix operations, and Python's ability to manipulate lists elegantly.

10. Detect a Cycle in a Linked List 🔁

FAANG loves linked lists! A cycle detection problem helps check your understanding of fast and slow pointers.

🔍 **Question**: Write a function to detect if a linked list has a cycle.

```
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def has_cycle(head: ListNode) -> bool:
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return True
    return False
```

Why it's important: It's all about pointer manipulation and working efficiently with linked data structures.

11. Find the Intersection of Two Linked Lists ⚓

Imagine two linked lists that eventually merge into a single list. Your task is to find the node at which these two lists intersect.

🔍 **Question**: Write a function to find the intersection of two singly linked lists.

```
class ListNode:
    def __init__(self, value=0, next=None):
```

```
        self.value = value
        self.next = next

def get_intersection_node(headA: ListNode, headB: ListNode) -> ListNode:
    pointerA, pointerB = headA, headB

    while pointerA != pointerB:
        pointerA = pointerA.next if pointerA else headB
        pointerB = pointerB.next if pointerB else headA

    return pointerA

# Example
# Assuming the lists intersect at node 8
# ListA: 4 -> 1 -> 8 -> 4 -> 5
# ListB: 5 -> 6 -> 1 -> 8 -> 4 -> 5
```

Why it's important: This problem focuses on manipulating linked lists and understanding pointer traversal, a critical skill for coding interviews.

12. Serialize and Deserialize a Binary Tree 🌳

This problem involves working with trees, a key data structure in FAANG interviews. **Serialization** means converting the tree into a string, while **deserialization** means rebuilding the tree from that string.

🔍 **Question:** Write a function to serialize and deserialize a binary tree.

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

def serialize(root: TreeNode) -> str:
    def helper(node):
        if not node:
            return "None,"
        return str(node.value) + "," + helper(node.left) + helper(node.right)

    return helper(root)

def deserialize(data: str) -> TreeNode:
    def helper(nodes):
        value = next(nodes)
        if value == "None":
            return None
        node = TreeNode(int(value))
        node.left = helper(nodes)
        node.right = helper(nodes)
        return node

    return helper(iter(data.split(',')))

# Example
# Original Tree:     1
#                  /   \
#                 2     3
```

```
#                      / \
#                     4   5
# Serialized String: "1,2,None,None,3,4,None,None,5,None,None,"
```

13. Generate Parentheses Combinations 🧩

This problem is a popular choice to test your recursive thinking and **backtracking** skills.

🔍 **Question:** Write a function to generate all combinations of well-formed parentheses given `n` pairs of parentheses.

```
def generate_parentheses(n: int) -> list:
    result = []

    def backtrack(s='', left=0, right=0):
        if len(s) == 2 * n:
            result.append(s)
            return
        if left < n:
            backtrack(s + '(', left + 1, right)
        if right < left:
            backtrack(s + ')', left, right + 1)

    backtrack()
    return result

# Example
print(generate_parentheses(3))
# Output: ['((()))', '(()())', '(())()', '()(())', '()()()']
```

Why it's important: This problem evaluates your ability to explore all possibilities (backtracking) while maintaining constraints.

14. Trapping Rain Water 🌧️

This is a very common dynamic programming problem that involves calculating how much rainwater can be trapped between bars.

🔍 **Question:** Given `n` non-negative integers representing an elevation map, compute how much water it can trap after raining.

```
def trap(height: list) -> int:
    if not height:
        return 0

    left, right = 0, len(height) - 1
    left_max, right_max = height[left], height[right]
```

```
    water_trapped = 0

    while left < right:
        if left_max < right_max:
            left += 1
            left_max = max(left_max, height[left])
            water_trapped += left_max - height[left]
        else:
            right -= 1
            right_max = max(right_max, height[right])
            water_trapped += right_max - height[right]

    return water_trapped

# Example
print(trap([0,1,0,2,1,0,1,3,2,1,2,1]))  # Output: 6
```

Why it's important: This is an excellent test of your problem-solving abilities and dynamic programming knowledge.

15. Palindrome Partitioning ✂️

Given a string, split it into the minimum number of substrings where each substring is a palindrome.

🔍 **Question**: Write a function to partition a string into palindrome substrings.

```
def partition(s: str) -> list:
    result = []

    def backtrack(path, start):
        if start == len(s):
            result.append(path[:])
            return

        for end in range(start, len(s)):
            if s[start:end + 1] == s[start:end + 1][::-1]:
                path.append(s[start:end + 1])
                backtrack(path, end + 1)
                path.pop()

    backtrack([], 0)
    return result

# Example
print(partition("aab"))
# Output: [['a', 'a', 'b'], ['aa', 'b']]
```

Why it's important: This question tests your knowledge of backtracking and recursion, key elements for handling complex problems.

16. Word Search in a Grid 🔍

You need to find if a word exists in a 2D board, where the word can be constructed by sequentially adjacent letters.

🔍 **Question:** Write a function to find a word in a 2D grid of letters.

```
def exist(board: list, word: str) -> bool:
    def dfs(i, j, k):
        if not (0 <= i < len(board) and 0 <= j < len(board[0])):
            return False
        if board[i][j] != word[k]:
            return False
        if k == len(word) - 1:
            return True

        tmp, board[i][j] = board[i][j], "/"
        found = dfs(i + 1, j, k + 1) or dfs(i - 1, j, k + 1) or dfs(i, j + 1, k
        board[i][j] = tmp
        return found

    for i in range(len(board)):
        for j in range(len(board[0])):
            if dfs(i, j, 0):
                return True
    return False

# Example
board = [
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
print(exist(board, "ABCCED"))  # Output: True
```

Why it's important: This problem tests grid traversal techniques and backtracking.

17. Find Median in a Data Stream 📊

This problem asks you to continuously find the median as new numbers are added to the stream.

🔍 **Question:** Design a data structure to find the median of a stream of integers.

```
import heapq

class MedianFinder:
    def __init__(self):
        self.small = []  # Max-heap (simulated by inverting values)
```

```
        self.large = []  # Min-heap

    def add_num(self, num: int):
        heapq.heappush(self.small, -num)
        heapq.heappush(self.large, -heapq.heappop(self.small))

        if len(self.small) < len(self.large):
            heapq.heappush(self.small, -heapq.heappop(self.large))

    def find_median(self) -> float:
        if len(self.small) > len(self.large):
            return -self.small[0]
        return (-self.small[0] + self.large[0]) / 2

# Example
mf = MedianFinder()
mf.add_num(1)
mf.add_num(2)
print(mf.find_median())  # Output: 1.5
mf.add_num(3)
print(mf.find_median())  # Output: 2.0
```

Why it's important: This requires understanding of **heap** data structures and efficient stream processing.

18. Design an LRU Cache 🚀

A Least Recently Used (LRU) cache evicts the least used items first when capacity is reached.

🔍 **Question**: Design and implement an LRU cache.

```
from collections import OrderedDict

class LRUCache:
    def __init__(self, capacity: int):
        self.cache = OrderedDict()
        self.capacity = capacity

    def get(self, key: int) -> int:
        if key not in self.cache:
            return -1
        # Move to end to show that it was recently used
        self.cache.move_to_end(key)
        return self.cache[key]

    def put(self, key: int, value: int) -> None:
        if key in self.cache:
            self.cache.move_to_end(key)
        self.cache[key] = value
        if len(self.cache) > self.capacity:
            # Remove the first key-value pair from OrderedDict
            self.cache.popitem(last=False)

# Example
lru_cache = LRUCache(2)
lru_cache.put(1, 1)   # cache is {1=1}
lru_cache.put(2, 2)   # cache is {1=1, 2=2}
```

```
print(lru_cache.get(1))  # returns 1
lru_cache.put(3, 3)   # evicts key 2, cache is {1=1, 3=3}
print(lru_cache.get(2))  # returns -1 (not found)
lru_cache.put(4, 4)   # evicts key 1, cache is {3=3, 4=4}
print(lru_cache.get(1))  # returns -1 (not found)
print(lru_cache.get(3))  # returns 3
print(lru_cache.get(4))  # returns 4
```

Why it's important: This problem tests your understanding of data structures, specifically how to implement efficient caching mechanisms.

19. Flatten a Nested List Iterator 📜

This problem checks how well you can handle nested structures and iterators.

🔍 **Question**: Implement an iterator that flattens a nested list of integers.

```
class NestedInteger:
    def __init__(self, value=None):
        self.value = value
        self.is_integer = isinstance(value, int)
        self.list = value if isinstance(value, list) else []

    def isInteger(self) -> bool:
        return self.is_integer

    def getInteger(self) -> int:
        return self.value

    def getList(self) -> list:
        return self.list

class NestedIterator:
    def __init__(self, nested_list: list):
        self.flattened = []
        self.index = 0
        self.flatten(nested_list)

    def flatten(self, nested_list):
        for item in nested_list:
            if item.isInteger():
                self.flattened.append(item.getInteger())
            else:
                self.flatten(item.getList())

    def next(self) -> int:
        result = self.flattened[self.index]
        self.index += 1
        return result

    def hasNext(self) -> bool:
        return self.index < len(self.flattened)

# Example
nested_list = [NestedInteger(1), NestedInteger([2, NestedInteger(3)]), NestedInt
iterator = NestedIterator(nested_list)
```

```
while iterator.hasNext():
    print(iterator.next())  # Output: 1, 2, 3, 4
```

Why it's important: This tests your ability to traverse and manipulate nested structures effectively.

20. Kth Largest Element in a Stream 📈

This problem evaluates how to efficiently find the kth largest element as new numbers are added to a stream.

🔍 **Question:** Design a class that manages a stream of numbers and can return the kth largest number at any time.

```
import heapq

class KthLargest:
    def __init__(self, k: int, nums: list):
        self.k = k
        self.min_heap = nums
        heapq.heapify(self.min_heap)
        while len(self.min_heap) > k:
            heapq.heappop(self.min_heap)

    def add(self, val: int) -> int:
        heapq.heappush(self.min_heap, val)
        if len(self.min_heap) > self.k:
            heapq.heappop(self.min_heap)
        return self.min_heap[0]

# Example
kthLargest = KthLargest(3, [4, 5, 8, 2])
print(kthLargest.add(3))  # Output: 4
print(kthLargest.add(5))  # Output: 5
print(kthLargest.add(10)) # Output: 5
print(kthLargest.add(9))  # Output: 8
print(kthLargest.add(4))  # Output: 8
```

Why it's important: This problem tests your understanding of heaps and dynamic data structures.

21. Longest Consecutive Sequence 📈

This problem tests your ability to work with arrays and hash sets effectively.

🔍 **Question:** Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

```python
def longest_consecutive(nums: list) -> int:
    num_set = set(nums)
    longest_streak = 0

    for num in num_set:
        if num - 1 not in num_set:  # Check if it's the start of a sequence
            current_num = num
            current_streak = 1

            while current_num + 1 in num_set:
                current_num += 1
                current_streak += 1

            longest_streak = max(longest_streak, current_streak)

    return longest_streak

# Example
print(longest_consecutive([100, 4, 200, 1, 3, 2]))  # Output: 4 (1, 2, 3, 4)
```

Why it's important: This question assesses your understanding of hash tables and array traversal.

22. Combination Sum 🔍

This problem checks your backtracking abilities and is often favored in interviews.

🔍 **Question**: Given an array of distinct integers and a target sum, find all unique combinations that sum up to the target.

```python
def combination_sum(candidates: list, target: int) -> list:
    result = []

    def backtrack(start, path, remaining):
        if remaining == 0:
            result.append(path)
            return
        if remaining < 0:
            return

        for i in range(start, len(candidates)):
            backtrack(i, path + [candidates[i]], remaining - candidates[i])

    backtrack(0, [], target)
    return result

# Example
print(combination_sum([2, 3, 6, 7], 7))
# Output: [[2, 2, 3], [7]]
```

Why it's important: This question tests your ability to generate combinations and use recursion effectively.

23. Number of Islands 🏝️

This problem involves graph traversal and is a favorite for testing knowledge of breadth-first search (BFS) or depth-first search (DFS).

🔍 **Question**: Given a 2D grid of '1's (land) and '0's (water), count the number of islands.

```
def num_islands(grid: list) -> int:
    if not grid:
        return 0

    def bfs(i, j):
        queue = [(i, j)]
        while queue:
            x, y = queue.pop(0)
            for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
                nx, ny = x + dx, y + dy
                if 0 <= nx < len(grid) and 0 <= ny < len(grid[0]) and grid[nx][n
                    grid[nx][ny] = '0'  # Mark as visited
                    queue.append((nx, ny))

    count = 0
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if grid[i][j] == '1':
                count += 1
                bfs(i, j)  # Start BFS to mark the whole island

    return count

# Example
grid = [
    ["1", "1", "0", "0", "0"],
    ["1", "1", "0", "0", "0"],
    ["0", "0", "1", "0", "0"],
    ["0", "0", "0", "1", "1"]
]
print(num_islands(grid))  # Output: 3
```

Why it's important: This question evaluates your skills in graph traversal and the ability to manage state during traversal.

24. Rotate Image 🔄

This problem checks your understanding of matrices and in-place manipulation.

🔍 **Question:** Given an n x n 2D matrix representing an image, rotate the image by 90 degrees clockwise.

```
def rotate(matrix: list) -> None:
    n = len(matrix)
    for i in range(n):
        for j in range(i, n):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]  # Transpose
    matrix.reverse()  # Reverse each row

# Example
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
rotate(matrix)
print(matrix)
# Output: [[7, 4, 1], [8, 5, 2], [9, 6, 3]]
```

Why it's important: This tests your ability to manipulate matrices and your understanding of in-place operations.

25. Course Schedule 📚

This problem focuses on graph theory and topological sorting, which is essential for understanding dependencies.

🔍 **Question:** Given the total number of courses and a list of prerequisites, determine if you can finish all courses.

```
from collections import defaultdict, deque

def can_finish(num_courses: int, prerequisites: list) -> bool:
    indegree = [0] * num_courses
    graph = defaultdict(list)

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

    queue = deque([i for i in range(num_courses) if indegree[i] == 0])

    count = 0
    while queue:
        course = queue.popleft()
        count += 1
        for neighbor in graph[course]:
            indegree[neighbor] -= 1
            if indegree[neighbor] == 0:
                queue.append(neighbor)

    return count == num_courses
```

```
# Example
print(can_finish(2, [[1, 0]]))  # Output: True
print(can_finish(2, [[1, 0], [0, 1]]))  # Output: False
```

Why it's important: This problem assesses your understanding of directed graphs and how to detect cycles.

26. Minimum Path Sum 🌍

This problem tests your dynamic programming skills and how to navigate grids.

🔍 **Question**: Given a grid filled with non-negative numbers, find a path from the top-left to the bottom-right which minimizes the sum of the numbers along the path.

```
def min_path_sum(grid: list) -> int:
    if not grid:
        return 0

    rows, cols = len(grid), len(grid[0])
    for i in range(rows):
        for j in range(cols):
            if i == 0 and j == 0:
                continue
            up = grid[i - 1][j] if i > 0 else float('inf')
            left = grid[i][j - 1] if j > 0 else float('inf')
            grid[i][j] += min(up, left)

    return grid[-1][-1]

# Example
grid = [
    [1, 3, 1],
    [1, 5, 1],
    [4, 2, 1]
]
print(min_path_sum(grid))  # Output: 7 (1 → 3 → 1 → 1 → 1)
```

Why it's important: This question tests your ability to use dynamic programming to optimize solutions.

27. Subsets 🎈

This problem tests your ability to generate all subsets of a given set, often using backtracking.

🔍 **Question:** Given an integer array, return all possible subsets (the power set).

```
def subsets(nums: list) -> list:
    result = []

    def backtrack(start, path):
        result.append(path)
        for i in range(start, len(nums)):
            backtrack(i + 1, path + [nums[i]])

    backtrack(0, [])
    return result

# Example
print(subsets([1, 2, 3]))
# Output: [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]
```

Why it's important: This problem evaluates your skills in generating combinations and using recursion effectively.

28. Validate Binary Search Tree 🌳

This problem tests your understanding of tree data structures and their properties.

🔍 **Question:** Given a binary tree, determine if it is a valid binary search tree (BST).

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

def is_valid_bst(root: TreeNode) -> bool:
    def helper(node, low=float('-inf'), high=float('inf')):
        if not node:
            return True
        if not (low < node.value < high):
            return False
        return helper(node.left, low, node.value) and helper(node.right, node.va

    return helper(root)

# Example
root = TreeNode(2)
root.left = TreeNode(1)
root.right = TreeNode(3)
print(is_valid_bst(root))  # Output: True
```

Why it's important: This question evaluates your understanding of tree properties and recursive traversal.

29. Merge Intervals 📅

This problem tests your ability to manage and manipulate intervals efficiently.

🔍 **Question**: Given a collection of intervals, merge all overlapping intervals.

```
def merge(intervals: list) -> list:
    if not intervals:
        return []

    intervals.sort(key=lambda x: x[0])
    merged = [intervals[0]]

    for current in intervals[1:]:
        last_merged = merged[-1]
        if current[0] <= last_merged[1]:  # Overlapping intervals
            last_merged[1] = max(last_merged[1], current[1])  # Merge
        else:
            merged.append(current)

    return merged

# Example
intervals = [[1, 3], [2, 6], [8, 10], [15, 18]]
print(merge(intervals))  # Output: [[1, 6], [8, 10], [15, 18]]
```

Why it's important: This question assesses your ability to handle overlapping intervals efficiently.

30. Find First and Last Position of Element in Sorted Array 🔍

This problem tests your knowledge of binary search algorithms.

🔍 **Question**: Given an array of integers sorted in ascending order, find the starting and ending position of a given target value.

```
def search_range(nums: list, target: int) -> list:
    def find_left(nums, target):
        left, right = 0, len(nums) - 1
        while left <= right:
            mid = (left + right) // 2
            if nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1
        return left
```

```
    def find_right(nums, target):
        left, right = 0, len(nums) - 1
        while left <= right:
            mid = (left + right) // 2
            if nums[mid] <= target:
                left = mid + 1
            else:
                right = mid - 1
        return right

    left_index = find_left(nums, target)
    right_index = find_right(nums, target)

    if left_index <= right_index:
        return [left_index, right_index]
    return [-1, -1]

# Example
print(search_range([5, 7, 7, 8, 8, 10], 8))  # Output: [3, 4]
print(search_range([5, 7, 7, 8, 8, 10], 6))  # Output: [-1, -1]
```

Why it's important: This problem tests your ability to use binary search effectively to find positions in sorted data.

Conclusion 🎉

Congratulations! You've now explored **30 of the most commonly asked coding interview questions** in Python at FAANG companies. Each of these questions not only challenges your coding skills but also enhances your understanding of algorithms and data structures.

Remember, the key to excelling in coding interviews is practice, persistence, and a problem-solving mindset. Keep pushing yourself, stay curious, and best of luck on your interview journey! You can do this! 💪💻✨

Thanks for Reading!

If you enjoyed this, follow me to never miss another article on data science guides, tricks and tips, life lessons, and more!

In Plain English 🚀

*Thank you for being a part of the **In Plain English** community! Before you go:*

- Be sure to **clap** and **follow** the writer 👏
- Follow us: X | LinkedIn | YouTube | Discord | Newsletter
- Visit our other platforms: CoFeed | Differ
- More content at PlainEnglish.io

Python Programming | Interview | Faang | Data Science | Data Analysis

10

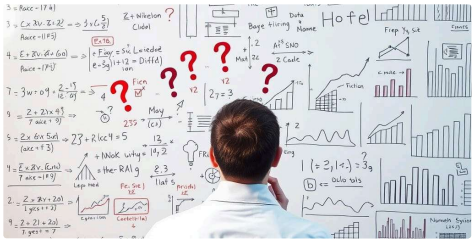
Written by Ritesh Gupta

Follow

3.5K Followers · Writer for Artificial Intelligence in Plain English

Data Scientist, I write Article on Machine Learning| Deep Learning| NLP | Open CV | AI Lover ❤️

More from Ritesh Gupta and Artificial Intelligence in Plain English

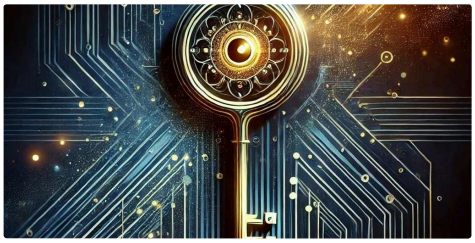


Ritesh Gupta

Can You Handle These 25 Toughest Data Science Interview Questions?

The role of a Data Scientist demands a unique blend of skills, including statistics, machine...

Sep 25 · 46

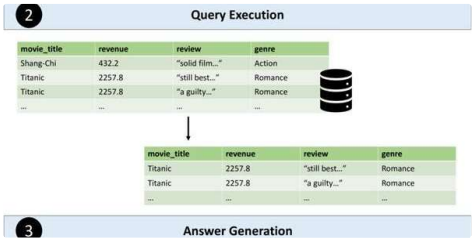


Andrew Be... in Artificial Intelligence in Plain Engli...

New KILLER ChatGPT Prompt—The "Playoff Method"

Super powerful prompt for ChatGPT—01 Preview

Sep 27 · 3.6K · 70



Pavan Ema... in Artificial Intelligence in Plain Engli...

Goodbye, Text2SQL: Why Table-Augmented Generation (TAG) is...

Ritesh Gupta

Meta's Llama 3.2: A Game-Changer in Generative AI

Exploring the Future of Natural Language Queries with Table-Augmented Generation.

Sep 11 1.2K 17

Generative AI continues to advance, and one of the most significant updates is the launch...

Sep 26 6

See all from Ritesh Gupta

See all from Artificial Intelligence in Plain English

Recommended from Medium

Nidhi Jain in Code Like A Girl

How art can transform your coding skills?

Boost Your Coding Skills with Creative Art

Sep 27 249 2

Rina Mondal

Data Science Interview Questions: Catch the Trend

Today, one of my students attended an interview for a Data Scientist role at a...

Sep 26 79 3

Lists

Predictive Modeling w/ Python

20 stories · 1610 saves

Coding & Development

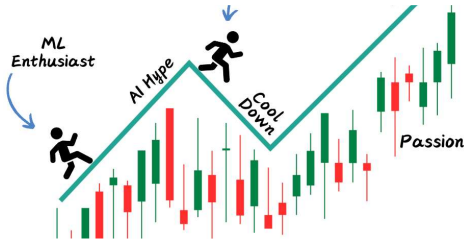
11 stories · 860 saves

Practical Guides to Machine Learning

10 stories · 1963 saves

ChatGPT prompts

50 stories · 2129 saves



Simranjeet Si... in Artificial Intelligence in Plain E...

Generative AI Interview Questions [LLM] Top 20—Part : 2

Crack you next Generative AI or LLM Interview with these Series of GenAI Intervie...

Aug 23 80

Kartik Singhal in Towards Data Science

Make the Switch from Software Engineer to ML Engineer

7 steps that helped me transition from a software engineer to Machine Learning...

Oct 8 650 8

Khushal Kumar

My Generative AI Engineer Interview Experience (Got Hired!)

I recently had a Gen AI interview at a tech company and these were the interview...

Oct 2 28

Mandy Liu in Level Up Coding

Become a Top 1% Candidate: My Behavioral Interview Playbook...

The R-STAR framework to crush “Tell me about a time when”

Sep 27 96

See more recommendations