Interview Guide

Top 45 Python Coding Interview Questions and Answers

October 25, 2024 Aseem

Do you know that Python ranks #1 in job postings across many sectors? Particularly in areas like data science, AI, machine learning, and DevOps. This demand for Python professionals is expected to grow due to its applications in emerging technologies such as AI, quantum computing, and IoT. Whether you are a newcomer excited to start your journey or a mid-level professional looking to elevate your career, preparing for Python coding interview questions is essential. Even for experienced developers aiming for higher-level roles, mastering these questions can significantly enhance your job prospects. In this blog, we have presented a collection of Python coding interview questions and answers tailored to different experience levels to help you prepare effectively to ace your upcoming interviews.

Table of Contents

Python Coding Interview Questions and Answers for Freshers

This section outlines common Python coding job interview questions and answers designed specifically for recent graduates. These questions aim to assess your foundational knowledge of Python, problem-solving skills, and understanding of essential programming concepts. Familiarize yourself with the following coding questions that can enhance your confidence and demonstrate your enthusiasm for the language, making a positive impression on potential employers.

Q1. Write a Python program to check if a number is even or odd.

Sample Answer: Here is a sample of how to write a Python program to check if a number is even or odd:

```
num = int(input("Enter a number: "))
if num % 2 == 0:
    print("Even")
else:
    print("Odd")
```

Q2. Write a Python function to reverse a string.

Sample Answer: This function, named reverse_string, takes a single string as input and uses Python's slicing feature to return the string in reverse order. Slicing allows for concise manipulation of strings, making it easy to reverse them with just one line of code.

Here is a sample of how to write a Python function to reverse a string.

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

Q3. How can you find the largest number in a list?

Sample Answer: To find the largest number in a list, you can define the function find_largest(numbers) to take a list as input and use the max() function to return the largest number from the list.

Here is an example of how you can do it:

```
def find_largest(numbers):
    return max(numbers)
```

In this code, max(numbers) scans through the list and returns the highest value, making it an effective solution for this problem.

Q4. Write a program to calculate the factorial of a number.

Sample Answer: The factorial of a number is defined as the product of all positive integers less than or equal to that number, and it is denoted by '*n!*'. A recursive approach is an effective way to calculate the factorial, where the function calls itself with a decremented value until it reaches the base case.

Here's how you can write a program to calculate the factorial of a number:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

The function factorial(n) checks if *n* is 0 (the base case), returning 1. Otherwise, it multiplies *n* by the factorial of n−1n – 1n−1, recursively computing the result until it reaches the base case.

Q5. How do you check if a string is a palindrome?

Sample Answer: A palindrome is a word or phrase that reads the same forwards and backward, ignoring spaces, punctuation, and capitalization. To check if a string is a palindrome, you can compare the original string with its reversed version. For example:

```
def is_palindrome(s):
    return s == s[::-1]
```

The function 'is_palindrome(s)' checks if the input string s is equal to its reverse, 's[::-1]'. If they are the same, the function returns 'True', indicating that the string is a palindrome. Otherwise, it returns 'False'.

Q6. Write a program to count the number of vowels in a given string.

Sample Answer: Vowels in the English language are ‘a’, ‘e’, ‘i’, ‘o’, and ‘u’. You can iterate over the string and count how many of these vowels it contains. In the following example, the function ‘count_vowels(s)’ use a generator expression to iterate through each character in the input string ‘s’, converting each character to lowercase to ensure the check is case-insensitive. The ‘sum()’ function then totals the number of vowels found.

Here’s the code:

```
def count_vowels(s):
    return sum(1 for char in s if char.lower() in 'aeiou')
```

Q7. How do you remove duplicates from a list?

Sample Answer: Removing duplicates from a list is a common task in data manipulation. One efficient way to achieve this is by converting the list to a set, as the sets automatically eliminate duplicate values. After removing duplicates, you can convert the set back to a list if needed.

Here’s how you can implement this approach:

```
def remove_duplicates(lst):
    return list(set(lst))
```

Q8. Write a program to find the common elements in two lists.

Sample Answer: Finding common elements between two lists is a useful operation in data analysis; one efficient way to do this is by using set operations, specifically the intersection. By converting both lists into sets, you can easily identify elements that appear in both.

Here’s how you can implement this:

```
def common_elements(list1, list2):
    return list(set(list1) & set(list2))
```

Q9. How do you merge two dictionaries in Python?

Sample Answer: When merging two dictionaries, if both dictionaries have the same key, the value from the second dictionary will overwrite the value from the first. Here is how you can merge two dictionaries in Python:

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
merged = {**dict1, **dict2}
```

Q10. Write a program to sort a list of numbers.

Sample Answer: Sorting a list can be easily done using Python’s built-in sorted() function. For example:

```
def sort_numbers(numbers):
    return sorted(numbers)
```

Q11. How can you count the occurrences of each character in a string?

Sample Answer: Counting the occurrences of each character in a string can be useful for various text analysis tasks, such as frequency analysis or data compression. A convenient way to achieve this in Python is by using the ‘collections.Counter’, which efficiently counts the frequency of each element in an iterable.

Here’s an example of how to implement this:

```
from collections import Counter
def count_characters(s):
    return dict(Counter(s))
```

Q12. Write a program to find the Fibonacci series up to n.

Sample Answer: The Fibonacci sequence starts with 0 and 1, and each subsequent number is the sum of the two preceding numbers. You can generate the sequence up to n terms. For example:

```
def fibonacci(n):
    fib_series = [0, 1]
    while len(fib_series) < n:
        fib_series.append(fib_series[-1] + fib_series[-2])
    return fib_series[:n]
```

Q13. Write a program to find the intersection of two sets.

Sample Answer: Set intersection finds the common elements between two sets. To find the intersection of two sets, you can write your code like this:

```
def intersection(set1, set2):
    return set1 & set2
```

Q14. How do you convert a list of strings to uppercase?

Sample Answer: To convert a list of strings to uppercase, you can iterate through the list and convert each string to uppercase using Python's str.upper() method. For example:

```
def to_uppercase(strings):
    return [s.upper() for s in strings]
```

Q15. How do you find the length of a list without using the len() function?

Sample Answer: Finding the length of a list without using the built-in 'len()' function is a useful exercise to understand iteration and counting in Python. You can achieve this by iterating through the list and incrementing a counter for each element encountered.

Here's how you can approach it:

```
def get_length(lst):
    count = 0
    for _ in lst:
        count += 1
    return count
```

Pro Tip: Python developers are paid a good amount at some of the top IT companies in India. You can learn about the Python developer salary in India and explore the salary packages as you grow in career.

Python Programming Interview Questions and Answers for Mid-level Candidates

This section focuses on Python programming interview questions designed for mid-level candidates. You will face questions that evaluate your practical use of Python, understanding of advanced concepts, and capability to solve real-world problems. By preparing for the following Python programming interview questions and answers, you can demonstrate your expertise and readiness for the next phase of your career.

Q16. Write a Python function to check if a number is prime.

Sample Answer: A prime number is defined as a number greater than 1 that has no divisors other than 1 and itself. To check if a number is prime, you can iterate from 2 up to the square root of the number, testing for any divisors. This method is efficient because if a number has a divisor larger than its square root, the corresponding quotient must be smaller than the square root.

Here is how you can implement the code:

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
```

Q17. Write a program to find the second largest number in a list.

Sample Answer: To find the second largest number, you can remove duplicates by converting the list to a set. Once you have a set of unique numbers, you can sort them in ascending order and retrieve the second-to-last element.

Here's how you can implement the code:

```
def second_largest(numbers):
    unique_numbers = list(set(numbers))
    unique_numbers.sort()
    return unique_numbers[-2] if len(unique_numbers) >= 2 else None
```

Q18. Write a Python function to flatten a nested list.

Sample Answer: To flatten a nested list, which is a list that may contain other lists as its elements, you can use a recursive approach. This method involves iterating through each item in the list and checking if it is itself a list. If it is, you recursively flatten that sublist; if it's not, you simply add it to the flattened list.

Here is an example of how to write a Python function to flatten a nested list:

```
def flatten(nested_list):
    flat_list = []
    for item in nested_list:
        if isinstance(item, list):
            flat_list.extend(flatten(item))
        else:
            flat_list.append(item)
    return flat_list
```

Q19. How can you implement a binary search in Python?

Sample Answer: Binary search is an efficient algorithm for finding a target in a sorted array. It repeatedly divides the search interval in half until the target is found.

Here is how to implement a binary search in Python:

```
def binary_search(arr, target):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] < target:
            low = mid + 1
        elif arr[mid] > target:
            high = mid - 1
        else:
            return mid
    return -1
```

Pro Tip: Dealing with different types of Python technical interview coding questions can be made easy with our guide on Python cheat sheet. It will help you understand how you can tackle coding logic effectively.

Q20. Write a program to check for an anagram.

Sample Answer: Two strings are considered anagrams if they can be rearranged to form one another. This means they contain the same characters in the same frequency but in a different order. A simple way to check for anagrams is to sort both strings and compare the sorted versions. If the sorted strings are identical, then the original strings are anagrams of each other. For example:

```
def are_anagrams(s1, s2):
    return sorted(s1) == sorted(s2)
```

Q21. How do you find the longest common prefix in a list of strings?

Sample Answer: The longest common prefix is the longest sequence of characters shared by all strings. You can find it by iteratively comparing each string with the prefix. For example:

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

Q22. Write a Python function to generate all permutations of a list.

Sample Answer: Permutations refer to all possible arrangements of the elements in a list, where the order of the elements matters. You can leverage Python's built-in 'itertools.permutations()' function to generate these permutations.

Here's how you can write a Python function to generate all permutations of a given list:

```
from itertools import permutations

def generate_permutations(lst):
    return list(permutations(lst))
```

Q23. How do you implement a stack using a list?

Sample Answer: A stack is a data structure that follows a last-in, first-out (LIFO) principle. The last item added to the stack is the first one to be removed. You can implement a stack in Python using lists, which allow you to define basic operations such as push (to add an item), pop (to remove the most recently added item), and check if the stack is empty.

Here's an example of how to create a stack class using a list:

```
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        return self.stack.pop() if not self.is_empty() else None

    def is_empty(self):
        return len(self.stack) == 0
```

Q24. Write a Python function to find the mode of a list.

Sample Answer: The mode of a list is defined as the element that appears most frequently within that list. To find the mode efficiently, you can use the Counter class from the collections module, which counts the frequency of each element.

Here's how you can implement a Python function to find the mode of a list:

```
from collections import Counter

def find_mode(numbers):
    count = Counter(numbers)
    mode_data = count.most_common()
    return mode_data[0][0]
```

Q25. How do you rotate a list to the right by k steps?

Sample Answer: Rotating a list to the right by *k* steps means shifting its elements to the right, such that the last *k* elements wrap around to the front of the list. This operation can be efficiently achieved using slicing in Python.

Here's how you can implement a function to rotate a list:

```
def rotate(lst, k):
    k = k % len(lst)
    return lst[-k:] + lst[:-k]
```

Q26. Write a function to check if a given string is a valid palindrome considering only alphanumeric characters.

Sample Answer: To check if a string is a valid palindrome while ignoring non-alphanumeric characters, you can clean the string using regular expressions and then compare it with its reverse. Here is an example of how to check if a given string is a valid palindrome considering only alphanumeric characters:

```
import re
def is_valid_palindrome(s):
    s = re.sub(r'[^a-zA-Z0-9]', '', s).lower()
    return s == s[::-1]
```

Q27. How do you calculate the power of a number using recursion?

Sample Answer: Calculating the power of a number recursively involves multiplying the base by itself a specified number of times, which is determined by the exponent. The recursion continues until the exponent reaches zero, at which point the result is 1, as any number raised to the power of zero is defined as 1.

Here's the code:

```
def power(base, exp):
    if exp == 0:
        return 1
    return base * power(base, exp - 1)
```

Q28. Write a Python function to find all unique triplets in a list that sum up to zero.

Sample Answer: To find triplets that sum to zero, you can sort the list and use a two-pointer approach to find combinations. Here is a sample Python function to find all unique triplets in a list that sum up to zero:

```
def three_sum(nums):
    nums.sort()
    result = []
    for i in range(len(nums) - 2):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        left, right = i + 1, len(nums) - 1
        while left < right:
            total = nums[i] + nums[left] + nums[right]
            if total < 0:
                left += 1
            elif total > 0:
                right -= 1
            else:
                result.append([nums[i], nums[left], nums[right]])
                left += 1
                right -= 1
                while left < right and nums[left] == nums[left - 1]:
                    left += 1
                while left < right and nums[right] == nums[right + 1]:
                    right -= 1
    return result
```

Pro Tip: Python developers can level up their careers by learning about full-stack web development. You can opt for our full stack development placement guarantee course and get assured job placement assistance.

Q29. How do you find the intersection of two sorted arrays?

Sample Answer: Finding the intersection of two sorted arrays involves identifying the common elements that appear in both arrays. This can be efficiently achieved using a two-pointer technique, where you maintain a pointer for each array and move them based on the comparison of the elements.

Here's the code to find the intersection of two sorted arrays:

```
def intersection_sorted(arr1, arr2):
    result = []
    i, j = 0, 0
    while i < len(arr1) and j < len(arr2):
        if arr1[i] < arr2[j]:
            i += 1
        elif arr1[i] > arr2[j]:
            j += 1
        else:
            result.append(arr1[i])
            i += 1
            j += 1
    return result
```

Q30. Write a Python function to compute the Least Common Multiple (LCM) of two numbers.

Sample Answer: The least common multiple (LCM) can be computed using the greatest common divisor (GCD), which you can get from Python's 'math.gcd()' function. Here is an example of how to write a Python function to compute the least common multiple (LCM) of two numbers:

```
from math import gcd
def lcm(a, b):
    return abs(a * b) // gcd(a, b)
```

Python Coding Interview Questions and Answers for Experienced Professionals

For experienced developers, Python coding interviews require a higher level of skill and understanding. This segment covers advanced Python coding interview questions for experienced professionals. The following questions will help you demonstrate your raw coding skills and understanding of best practices, design patterns, and performance optimization.

Q31. Write a Python function to find the longest increasing subsequence in a list.

Sample Answer: The longest increasing subsequence (LIS) problem involves finding the length of the longest subsequence where each element is greater than the previous one. This can be solved using dynamic programming.

Here is the code for finding the longest increasing subsequence in a list:

```
def longest_increasing_subsequence(nums):
    if not nums:
        return 0
    dp = [1] * len(nums)
    for i in range(1, len(nums)):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] = max(dp[i], dp[j] + 1)
    return max(dp)
```

Q32. How do you implement a binary tree and perform an in-order traversal?

Sample Answer: A binary tree is a hierarchical structure in which each node can have at most two children, referred to as the left and right child. In-order traversal is a depth-first traversal method that visits the left subtree, the root node, and then the right subtree recursively, resulting in the nodes being processed in ascending order for a binary search tree.

Here's how you can implement a binary tree and perform an in-order traversal in Python:

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.value = key

def in_order_traversal(root):
    if root:
        in_order_traversal(root.left)               #visit the left subtree
        print(root.value, end=' ')                    #visit root
        in_order_traversal(root.right)             #visit right subtree
```

Q33. Write a function to find all valid parentheses combinations for a given number of pairs.

Sample Answer: Generating all valid combinations of parentheses for a given number of pairs can be approached using backtracking. It systematically explores all possible combinations while ensuring they remain valid. In this case, valid combinations must have matching opening and closing parentheses. Here's how you can code a function to find all valid parentheses combinations for *n* pairs:

```
def generate_parentheses(n):
    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)
    result = []
    backtrack()
    return result
```

Q34. How do you implement a queue using two stacks?

Sample Answer: A queue follows a first-in, first-out (FIFO) principle, while stacks operate on a last-in, first-out (LIFO) basis. You can implement a queue using two stacks: one for handling enqueue operations and the other for managing dequeue operations. This method effectively simulates the queue's behavior by reversing the order of elements when necessary.

Here's how you can code a queue using two stacks in Python:

```
class Queue:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def enqueue(self, item):
        self.stack1.append(item)

    def dequeue(self):
        if not self.stack2:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        return self.stack2.pop() if self.stack2 else None
```

Q35. Write a Python function to find the maximum product of two integers in a list.

Sample Answer: To find the maximum product of two integers in a list, you need to consider both the two largest numbers and the two smallest numbers (in the case of negative integers, their product can be larger). This approach ensures that you capture all potential pairs that could yield the highest product.

Here's how you can write a Python function to find the maximum product of two integers in a given list:

```
def max_product(lst):
if len(lst) < 2:
return None                    # Not enough elements to form a product

# Sort the list to easily access the largest and smallest values
lst.sort()

# Calculate the product of the two largest and two smallest numbers
return max(lst[-1] * lst[-2], lst[0] * lst[1])
```

Q36. How do you find the median of a list in linear time?

Sample Answer: Using the quickselect algorithm, similar to quicksort, you can find the kth smallest element in linear time on average. This can be used to compute the median. For example:

```
import random

def quickselect(arr, k):
    if len(arr) == 1:
        return arr[0]
    pivot = random.choice(arr)
    lows = [x for x in arr if x < pivot]
    highs = [x for x in arr if x > pivot]
    pivots = [x for x in arr if x == pivot]
    if k < len(lows):
        return quickselect(lows, k)
    elif k < len(lows) + len(pivots):
        return pivots[0]
    else:
        return quickselect(highs, k - len(lows) - len(pivots))

def find_median(nums):
    n = len(nums)
    if n % 2 == 1:
        return quickselect(nums, n // 2)
    else:
        return (quickselect(nums, n // 2 - 1) + quickselect(nums, n // 2)) / 2
```

Q37. Write a Python function to solve the N-Queens problem.

Sample Answer: The N-Queens problem involves placing N queens on an N×N chessboard such that no two queens attack each other. This can be solved using backtracking.

Here is an example of how to write a Python function to solve the N-Queens Problem:

```
def solve_n_queens(n):
    def backtrack(row, cols, diagonals1, diagonals2):
        if row == n:
            result.append(board[:])
            return
        for col in range(n):
            if col in cols or (row - col) in diagonals1 or (row + col) in diagonals2:
                continue
            board[row] = col
            cols.add(col)
            diagonals1.add(row - col)
            diagonals2.add(row + col)
            backtrack(row + 1, cols, diagonals1, diagonals2)
            cols.remove(col)
            diagonals1.remove(row - col)
            diagonals2.remove(row + col)

    result = []
    board = [-1] * n
    backtrack(0, set(), set(), set())
    return result
```

Q38. How can you implement Dijkstra's algorithm for finding the shortest path in a graph?

Sample Answer: Dijkstra's algorithm is used to find the shortest path from a starting node to all other nodes in a weighted graph. Here's how you can implement it:

```
import heapq

def dijkstra(graph, start):
    min_heap = [(0, start)]
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    while min_heap:
        current_distance, current_node = heapq.heappop(min_heap)
        if current_distance > distances[current_node]:
            continue
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(min_heap, (distance, neighbor))
    return distances
```

Q39. Write a Python function to merge k-sorted linked lists.

Sample Answer: To merge k-sorted linked lists, you can use a min-heap to always extract the smallest element from each list. Here's a code to merge k-sorted linked lists:

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

def merge_k_lists(lists):
    import heapq
    min_heap = []
    for l in lists:
        if l:
            heapq.heappush(min_heap, (l.value, l))

    dummy = ListNode()
    current = dummy
    while min_heap:
        value, node = heapq.heappop(min_heap)
        current.next = node
        current = current.next
        if node.next:
            heapq.heappush(min_heap, (node.next.value, node.next))

    return dummy.next
```

Q40. How do you implement a trie data structure?

Sample Answer: A trie is a tree-like structure that helps store strings efficiently, where each node represents a single character. Below is an example of how to implement a trie:

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True

    def search(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                return False
            node = node.children[char]
        return node.is_end_of_word

    def starts_with(self, prefix):
        node = self.root
        for char in prefix:
            if char not in node.children:
                return False
            node = node.children[char]
        return True
```

Q41. Write a function to find the longest palindromic substring in a string.

Sample Answer: You can expand around each character and its neighbor to check for palindromes to find the longest palindromic substring in a string. This method explores all possible centers of the palindrome.

Here's how you can implement this:

```
def longest_palindrome(s):
    def expand_from_center(left, right):
        while left >= 0 and right < len(s) and s[left] == s[right]:
            left -= 1
            right += 1
        return right - left - 1

    start, end = 0, 0
    for i in range(len(s)):
        len1 = expand_from_center(i, i)
        len2 = expand_from_center(i, i + 1)
        max_len = max(len1, len2)
        if max_len > end - start:
            start = i - (max_len - 1) // 2
            end = i + max_len // 2
    return s[start:end + 1
```

Q42. How do you find all combinations of a given sum in a list?

Sample Answer: To find all combinations of numbers that sum up to a target value, you can use backtracking. Here is how you can find all combinations of a given sum in a list:

```
def combination_sum(candidates, target):
    def backtrack(start, path, total):
        if total == target:
            result.append(path)
            return
        if total > target:
            return
        for i in range(start, len(candidates)):
            backtrack(i, path + [candidates[i]], total + candidates[i])

    result = []
    backtrack(0, [], 0)
    return result
```

Q43. Write a function to find the shortest path in a maze using BFS.

Sample Answer: To find the shortest path in a maze, use the breadth-first search (BFS) algorithm. This method explores all possible paths level by level, ensuring that the shortest path from the starting point to the endpoint is found.

Here's how you can implement this:

```
from collections import deque

def shortest_path(maze, start, end):
    rows, cols = len(maze), len(maze[0])
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    queue = deque([(start, 0)])
    visited = set()
    visited.add(start)

    while queue:
        (x, y), steps = queue.popleft()
        if (x, y) == end:
            return steps

        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < rows and 0 <= ny < cols and maze[nx][ny] == 0 and (nx, ny) not in visited:
                visited.add((nx, ny))
                queue.append(((nx, ny), steps + 1))
    return -1  # return -1 if no path is found
```

Q44. How do you detect a cycle in a directed graph using DFS?

Sample Answer: To detect cycle in a directed graph, you can use depth-first search (DFS). You keep track of the nodes currently being explored, and if you visit a node that's already in this exploration path, it means there's a cycle. Here's how you can implement this:

```
def has_cycle(graph):
    def dfs(node):
        if node in visiting:
            return True
        if node in visited:
            return False
        visiting.add(node)
        for neighbor in graph[node]:
            if dfs(neighbor):
                return True
        visiting.remove(node)
        visited.add(node)
        return False

    visited, visiting = set(), set()
    for vertex in graph:
        if dfs(vertex):
            return True
    return False
```

Q45. Write a function to find the longest substring without repeating characters.

Sample Answer: You can use a method called the sliding window technique, which helps keep track of a range of unique characters. The window is adjusted when a repeated character is found, and you keep track of the longest length of this window to determine the longest substring without repeating characters.

Here's how you can write the code to find that substring without repeating characters:

```
def length_of_longest_substring(s):
    char_index = {}
    max_length = start = 0
    for index, char in enumerate(s):
        if char in char_index and char_index[char] >= start:
            start = char_index[char] + 1
        char_index[char] = index
        max_length = max(max_length, index - start + 1)
    return max_length
```


Conclusion

Whether you are a recent graduate or senior candidate, the key to managing coding and programming interviews is to step up your coding practice. Each phase of your career presents unique challenges and knowing what is expected at your level of experience can greatly enhance your chances of success. By going through these carefully selected Python coding interview questions and answers, you can refine your skills, boost your confidence, and present yourself as a strong candidate. To expand your career horizon, explore our curated collection of 45 essential TCS Python interview questions to equip yourself with the knowledge you need to stand out and succeed.

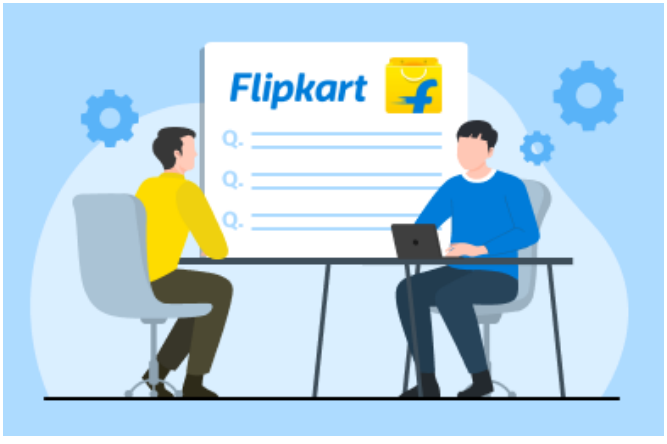
← Previous

Next →

Aseem

A seasoned tech professional, Aseem Garg is Internshala's Vice President of Engineering. A Full Stack Web Engineer and Android Engineer, he is responsible for leading and driving innovative technology at Internshala. With nine years of rich experience, he is an innovator - passionate about creating seamless web and mobile experiences while implementing efficient DevOps practices.

Related Post

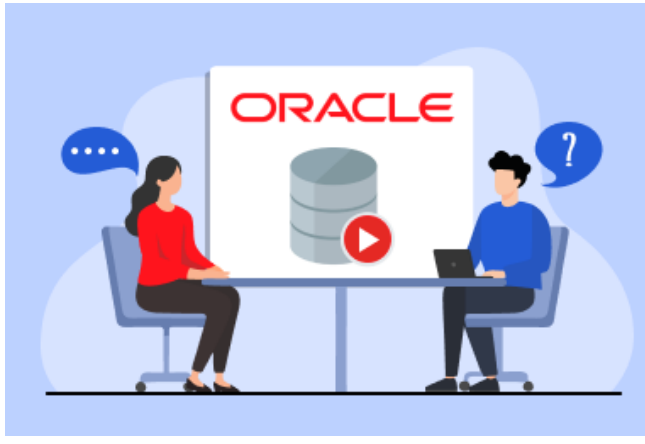


Interview Guide

Top 40 Flipkart BPO Interview Questions and Answers: With Intervie...

Flipkart, one of India's leading e-commerce websites, frequently recruits for various roles in its Business Process Outsourcing or BPO division. These roles... include telesales executives, customer care

April 4, 2025 Surbhi Girdhar



Interview Guide

Top 40 Oracle SQL Interview Questions and Answers: Your Guide to Success

Preparing for the Oracle SQL interview questions starts with mastering key concepts, practicing common queries, and understanding advanced... database operations. Oracle SQL is a critical skill for

April 4, 2025 Jahanvi Rana