Frequently Asked Python Interview Questions

Difficulty level: Easy

1. Write a Python program to swap two variables.

```
a = 10
b = 20
a, b = b, a
print("Swapped values: a =", a, "b =", b)
```

2. Create a function that returns the factorial of a number.

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

num = 5
print("Factorial of", num, "is", factorial(num))
```

3. Write a program to check if a number is prime.

```
def is_prime(num):

    if num < 2:
        return False

    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

number = 17
if is_prime(number):
    print(number, "is a prime number")
else:
    print(number, "is not a prime number")
```

4. Implement a function to reverse a string.

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

```
text = "hello"
print("Reversed string:", reverse_string(text))
```

5. Write a Python program to find the largest element in a list.

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

numbers = [10, 5, 8, 12, 7]
largest = find_largest(numbers)
print("Largest element:", largest)
```

6. Create a function to check if a number is even or odd.

```
def is_even(num):
    return num % 2 == 0

number = 7
if is_even(number):
    print(number, "is even")
else:
    print(number, "is odd")
```

7. Write a program to find the sum of all elements in a list.

```
def sum_of_elements(lst):
    return sum(lst)

numbers = [2, 4, 6, 8, 10]
total = sum_of_elements(numbers)
print("Sum of elements:", total)
```

8. Implement a function to check if a string is a palindrome.

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

word = "racecar"
if is_palindrome(word):
    print(word, "is a palindrome")
else:
    print(word, "is not a palindrome")
```

9. Write a Python program to find the average of a list of numbers.

```
def calculate_average(lst):
    return sum(lst) / len(lst)

numbers = [5, 10, 15, 20, 25]
average = calculate_average(numbers)
print("Average:", average)
```

10. Create a function to convert Celsius to Fahrenheit.

```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

temperature = 25
fahrenheit = celsius_to_fahrenheit(temperature)
print(temperature, "°C is equal to", fahrenheit, "°F")
```

11. Write a program to find the second largest element in a list.

```
def find_second_largest(lst):
    unique_elements = set(lst)
    unique_elements.remove(max(unique_elements))
    return max(unique_elements)

numbers = [10, 5, 8, 12, 12, 7]
second_largest = find_second_largest(numbers)
print("Second largest element:", second_largest)
```

12. Implement a function to count the number of vowels in a string.

```
def count_vowels(s):
    vowels = 'aeiou'
    count = 0
    for char in s.lower():
        if char in vowels:
            count += 1
    return count

text = "Hello, World!"
```

```
num_vowels = count_vowels(text)
print("Number of vowels:", num_vowels)
```

13. Write a Python program to find the product of all elements in a list.

```
def product_of_elements(lst):
    result = 1
    for num in lst:
        result *= num
    return result

numbers = [2, 3, 4, 5]
product = product_of_elements(numbers)
print("Product of elements:", product)
```

14. Create a function to check if a number is a perfect square.

```
import math

def is_perfect_square(num):
    sqrt = math.sqrt(num)
    return sqrt.is_integer()

number = 16
if is_perfect_square(number):
    print(number, "is a perfect square")
else:
    print(number, "is not a perfect square")
```

15. Write a program to find the common elements between two lists.

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

numbers1 = [1, 2, 3, 4, 5]
numbers2 = [4, 5, 6, 7, 8]
common = find_common_elements(numbers1, numbers2)
print("Common elements:", common)
```

16. Implement a function to remove duplicates from a list.

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

numbers = [1, 2, 3, 2, 4, 5, 1]
```

```
unique_numbers = remove_duplicates(numbers)
print("List without duplicates:", unique_numbers)
```

17. Write a Python program to find the maximum and minimum elements in a list.

```
def find_max_min(lst):
    return max(lst), min(lst)

numbers = [10, 5, 8, 12, 7]
max_value, min_value = find_max_min(numbers)
print("Maximum element:", max_value)
print("Minimum element:", min_value)
```

18. Create a function to check if a string is a valid email address.

```
import re

def is_valid_email(email):
    pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
    return bool(re.match(pattern, email))

email_address = "example@example.com"
if is_valid_email(email_address):
    print("Valid email address")
else:
    print("Invalid email address")
```

19. Write a program to find the sum of digits in a number.

```
def sum_of_digits(num):
    return sum(int(digit) for digit in str(num))

number = 12345
total = sum_of_digits(number)
print("Sum of digits:", total)
```

20. Implement a function to generate the first n Fibonacci numbers.

```
def fibonacci(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
```

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

num_terms = 10
fib_sequence = fibonacci(num_terms)
print("First", num_terms, "Fibonacci numbers:", fib_sequence)
```

Difficulty level: Medium

1. Write a program to find the Fibonacci series up to n terms.

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

num_terms = 10
fib_sequence = fibonacci(num_terms)
print("Fibonacci series up to", num_terms, "terms:", fib_sequence)
```

2. Create a Python class for a simple calculator.

```
class Calculator:
    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b

    def multiply(self, a, b):
        return a * b

    def divide(self, a, b):
        if b == 0:
            return "Error: Division by zero"
        return a / b

calc = Calculator()
print("5 + 3 =", calc.add(5, 3))
print("10 - 4 =", calc.subtract(10, 4))
print("2 * 6 =", calc.multiply(2, 6))
print("15 / 3 =", calc.divide(15, 3))
```

3. Write a program to sort a list of elements using the bubble sort algorithm.

```
def bubble_sort(lst):
    n = len(lst)

    for i in range(n):
        for j in range(0, n-i-1):
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]
    return lst

numbers = [5, 2, 8, 1, 9]
sorted_numbers = bubble_sort(numbers)
print("Sorted list:", sorted_numbers)
```

4**. Implement a function to find the intersection of two lists.**

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

numbers1 = [1, 2, 3, 4, 5]
numbers2 = [4, 5, 6, 7, 8]

intersection = find_intersection(numbers1, numbers2)
print("Intersection:", intersection)
```

5. Write a Python program to find the maximum element in a list using recursion.

```
def find_max(lst):
    if len(lst) == 1:
        return lst[0]
    else:
        return max(lst[0], find_max(lst[1:]))

numbers = [10, 5, 8, 12, 7]
max_element = find_max(numbers)
print("Maximum element:", max_element)
```

7. Create a function to reverse a list in-place.

```
def reverse_list(lst):
    left = 0
    right = len(lst) - 1

    while left < right:
        lst[left], lst[right] = lst[right], lst[left]
        left += 1
```

```
        right -= 1

    return lst

my_list = [1, 2, 3, 4, 5]
reversed_list = reverse_list(my_list)
print("Reversed list:", reversed_list)
```

8. Write a program to implement a queue using a list.

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

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

    def dequeue(self):
        if not self.is_empty():
            return self.queue.pop(0)
        else:
            return "Queue is empty"

    def is_empty(self):
        return len(self.queue) == 0

    def __len__(self):
        return len(self.queue)

queue = Queue()
queue.enqueue(10)
queue.enqueue(20)
queue.enqueue(30)
print("Dequeued element:", queue.dequeue())
print("Queue size:", len(queue))
```

10. Write a Python program to find the sum of all even numbers in a list.

```
def sum_of_even(lst):
    return sum(num for num in lst if num % 2 == 0)

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum = sum_of_even(numbers)
print("Sum of even numbers:", even_sum)
```

11. Create a function to find the length of the longest word in a string.

```
def longest_word_length(text):
    words = text.split()
    lengths = [len(word) for word in words]
    return max(lengths)

sentence = "The quick brown fox jumps over the lazy dog."
max_length = longest_word_length(sentence)
print("Length of the longest word:", max_length)
```

12. Write a program to implement a binary search algorithm.

```
def binary_search(lst, target):
    left = 0
    right = len(lst) - 1

    while left <= right:
        mid = (left + right) // 2
        if lst[mid] == target:
            return mid
        elif lst[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

numbers = [1, 3, 5, 7, 9]
print("Index of 5:", binary_search(numbers, 5))
print("Index of 2:", binary_search(numbers, 2))
```

13. Implement a function to find the number of occurrences of a character in a string.

```
def count_char(s, char):
    return s.count(char)

text = "Hello, World!"
count = count_char(text, "l")
print("Number of occurrences of 'l':", count)
```

14. Write a Python program to find the sum of all prime numbers in a list.

```
def is_prime(num):
    if num < 2:
```

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

def sum_of_primes(lst):
    return sum(num for num in lst if is_prime(num))

numbers = [2, 3, 4, 5, 6, 7, 8, 9, 10]
prime_sum = sum_of_primes(numbers)
print("Sum of prime numbers:", prime_sum)
```

15. Write a program to implement a stack using a list.

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

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

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        else:
            return "Stack is empty"

    def peek(self):
        if not self.is_empty():
            return self.stack[-1]
        else:
            return "Stack is empty"

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

    def __len__(self):
        return len(self.stack)

stack = Stack()
stack.push(10)
stack.push(20)
stack.push(30)
print("Top element:", stack.peek())
print("Popped element:", stack.pop())
print("Stack size:", len(stack))
```

16. Implement a function to find the missing number in a list of consecutive numbers.

```
def find_missing_number(lst):
    n = len(lst) + 1
    expected_sum = n * (n + 1) // 2
    actual_sum = sum(lst)
    return expected_sum - actual_sum

numbers = [1, 2, 3, 4, 6, 7, 8, 9, 10]
missing_number = find_missing_number(numbers)
print("Missing number:", missing_number)
```

17. Create a function to find the longest common prefix among a list of strings.

```
def longest_common_prefix(strs):
    if not strs:
        return ""

    prefix = strs[0]
    for i in range(1, len(strs)):
        while prefix and not strs[i].startswith(prefix):
            prefix = prefix[:-1]
        if not prefix:
            return ""

    return prefix

words = ["flower", "flow", "flight"]
common_prefix = longest_common_prefix(words)
print("Longest common prefix:", common_prefix)
```

18. Write a program to implement a linked list data structure.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
```

```
        if self.head is None:
            self.head = new_node
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def print_list(self):
        current_node = self.head
        while current_node:
            print(current_node.data)
            current_node = current_node.next

linked_list = LinkedList()
linked_list.append(1)
linked_list.append(2)
linked_list.append(3)
linked_list.print_list()
```

Difficulty level: Hard

1. Two Sum

Question
Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

Test Case
- Input: nums = [2,7,11,15], target = 9
- Output: [0,1]

Explanation
In this example, `nums[0] + nums[1] = 2 + 7 = 9`, so the output is `[0,1]`.

Solution

```
def two_sum(nums, target):
    num_map = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_map:
            return [num_map[complement], i]
        num_map[num] = i
```

2. Longest Substring Without Repeating Characters

Question
Given a string `s`, find the length of the longest substring without repeating characters.

Test Case
- Input: `s = "abcabcbb"`
- Output: `3`

Explanation
In this example, the longest substring without repeating characters is `"abc"`, which has a length of `3`.

Solution Explanation
We can use a sliding window approach to find the longest substring without repeating characters. We maintain a dictionary to store the index of the last occurrence of each character. We also maintain two pointers `start` and `end` to define the current substring. As we iterate through the string, if we encounter a character that is already in the dictionary, we update the `start` pointer to the next index after the last occurrence of that character. We update the dictionary with the current index of the character. At each step, we update the maximum length of the substring.

```
def length_of_longest_substring(s):
    char_map = {}
    start = 0
    max_length = 0
    for end in range(len(s)):
        if s[end] in char_map and char_map[s[end]] >= start:
            start = char_map[s[end]] + 1
        char_map[s[end]] = end
        max_length = max(max_length, end - start + 1)
    return max_length
```

Test cases:

```
print(length_of_longest_substring("abcabcbb"))  # Output: 3
print(length_of_longest_substring("bbbbb"))     # Output: 1
print(length_of_longest_substring("pwwkew"))    # Output: 3
print(length_of_longest_substring(""))          # Output: 0
```

3. Merge Intervals

Question
Given a collection of intervals, merge all overlapping intervals.

Test Case
- Input: `intervals = [[1,3],[2,6],[8,10],[15,18]]`
- Output: `[[1,6],[8,10],[15,18]]`

Explanation
In this example, the intervals `[1,3]` and `[2,6]` overlap, so they should be merged into `[1,6]`.

Solution Explanation
To merge overlapping intervals, we first sort the intervals based on their start times. Then, we iterate through the sorted intervals and merge overlapping intervals by comparing the end time of the current interval with the start time of the next interval. If there is an overlap, we merge the intervals by updating the end time of the current interval. If there is no overlap, we add the current interval to the result and update the current interval to the next interval.

```
def merge_intervals(intervals):
    if not intervals:
        return []
    intervals.sort(key=lambda x: x[0])
    merged = [intervals[0]]
    for interval in intervals[1:]:
        if interval[0] <= merged[-1][1]:
            merged[-1][1] = max(merged[-1][1], interval[1])
        else:
            merged.append(interval)
    return merged
```

Test cases:

```
print(merge_intervals([[1,3],[2,6],[8,10],[15,18]]))  # Output: [[1, 6], [8, 10], [15, 18]]
print(merge_intervals([[1,4],[4,5]]))                 # Output: [[1, 5]]
print(merge_intervals([[1,4],[0,2],[3,5]]))           # Output: [[0, 5]]
```

4. Top K Frequent Elements

Question
Given an integer array `nums` and an integer `k`, return the `k` most frequent elements. You may return the answer in any order.

Test Case
- Input: `nums = [1,1,1,2,2,3], k = 2`
- Output: `[1,2]`

Explanation
In this example, the two most frequent elements are `1` and `2`.

Solution Explanation
To find the `k` most frequent elements, we first count the frequency of each element using a dictionary. Then, we create a list of tuples where each tuple contains the element and its frequency. We sort this list based on the frequency in descending order. Finally, we extract the first `k` elements from the sorted list and return their values.

```
def top_k_frequent(nums, k):
    freq_map = {}
    for num in nums:
        freq_map[num] = freq_map.get(num, 0) + 1
    sorted_freq = sorted(freq_map.items(), key=lambda x: x[1], reverse=True)
    return [num for num, _ in sorted_freq[:k]]
```

Test cases:

```
print(top_k_frequent([1,1,1,2,2,3], 2))  # Output: [1, 2]
print(top_k_frequent([1], 1))            # Output: [1]
print(top_k_frequent([1,2,3,4,4,4,5,5,5], 3))  # Output: [4, 5, 1] (or [4, 5, 2] or [4, 5, 3])
```

5 .Rotate Array

Question
Given an array `nums` and an integer `k`, rotate the array to the right by `k` steps.

Test Case
- Input: `nums = [1,2,3,4,5,6,7], k = 3`
- Output: `[5,6,7,1,2,3,4]`

Explanation
In this example, after rotating the array to the right by 3 steps, the array becomes `[5,6,7,1,2,3,4]`.

Solution Explanation
To rotate the array to the right by `k` steps, we reverse the entire array, then reverse the first `k` elements, and finally reverse the rest of the elements. This effectively rotates the array to the right by `k` steps.

```
def rotate(nums, k):
    k %= len(nums)
    nums.reverse()
```

```
    nums[:k] = reversed(nums[:k])
    nums[k:] = reversed(nums[k:])
```

Test cases:

```
nums1 = [1,2,3,4,5,6,7]
rotate(nums1, 3)
print(nums1)  # Output: [5,6,7,1,2,3,4]

nums2 = [-1,-100,3,99]
rotate(nums2, 2)
print(nums2)  # Output: [3,99,-1,-100]

nums3 = [1,2]
rotate(nums3, 3)
print(nums3)  # Output: [2,1]
```

6. Minimum Window Substring

Question
Given two strings `s` and `t`, return the minimum window in `s` which contains all the characters in `t`. If there is no such window in `s` that covers all characters in `t`, return an empty string `""`.

Test Case
- Input: `s = "ADOBECODEBANC", t = "ABC"`
- Output: `"BANC"`

Explanation
In this example, the minimum window substring that contains all characters from string `t` is `"BANC"`. The substring starts at index 9 (character 'B') and ends at index 12 (character 'C') in the input string `s = "ADOBECODEBANC"`. This substring includes all characters 'A', 'B', and 'C' from string `t = "ABC"` in the correct order.

Solution Explanation
We can use a sliding window approach to find the minimum window substring. We maintain two pointers, `left` and `right`, to define the current window. We also maintain a dictionary `t_count` to store the count of characters in string `t`. As we expand the window by moving the `right` pointer, we decrease the count of the character in `t_count`. When all characters in `t` are found in the current window, we try to shrink the window from the left to minimize its size while still maintaining all

characters from `t`. We update the minimum window size and indices as we shrink the window.

```
def min_window(s, t):
    if not s or not t:
        return ""

    t_count = {}
    for char in t:
        t_count[char] = t_count.get(char, 0) + 1

    required_chars = len(t_count)
    left, right = 0, 0
    min_window_size = float("inf")
    min_window_start = 0
    char_count = {}

    while right < len(s):
        char = s[right]
        char_count[char] = char_count.get(char, 0) + 1
        if char in t_count and char_count[char] == t_count[char]:
            required_chars -= 1

        while required_chars == 0:
            if right - left + 1 < min_window_size:
                min_window_size = right - left + 1
                min_window_start = left

            char_left = s[left]
            char_count[char_left] -= 1
            if char_left in t_count and char_count[char_left] < t_count[char_left]:
                required_chars += 1

            left += 1

        right += 1

    return "" if min_window_size == float("inf") else
s[min_window_start:min_window_start+min_window_size]
```

Test cases:

```
print(min_window("ADOBECODEBANC", "ABC"))  # Output: "BANC"
print(min_window("a", "a"))                # Output: "a"
print(min_window("a", "aa"))               # Output: ""
```