TIP101 | Intro to Technical Interview Prep

Intro to Technical Interview Prep Fall 2025 (@ Section 3 | Tuesdays and Thursdays 5PM - 7PM PT)
Personal Member ID#: **134071**

Need help? Post on our [class slack channel]() or email us at [support@codepath.org]()

Session 2: Review II

Overview

Review of different topics covered in units 1-9. Be prepared to answer from a variety of different types of questions, invoking different algorithms and using different data structures.

> You can find all resources from today including the session deck, session recording, and more on the resources tab

🎢 Part 1 : Instructor Lead Session

We'll spend the first portion of the synchronous class time in large groups, where the instructor will lead class instruction for 30-45 minutes.

🧑‍💻 Part 2: Breakout Session

In breakout sessions, we will explore and collaboratively solve problem sets in small groups. Here, the **collaboration, conversation, and approach** are just as important as "solving the problem" - please engage warmly, clearly, and plentifully in the process!

In breakout rooms you will:

- Screen-share the problem/s, and verbally review them together
- Screen-share an interactive coding environment, and talk through the steps of a solution approach
  - ProTip: - An Integrated Development Environment (IDE) is a fancy name for a tool you could use for shared writing of code - like Replit.com, Collabed.it, CodePen.io, or other - your staff team will specify which tool to use for this class!
- Screen-share an implementation of your proposed solution
- Independently follow-along, or create an implementation, in your own IDE.

Your program leader/s will indicate which code sharing tool/s to use as a group, and will help break down or provide specific scaffolding with the main concepts above.

▶ **Note on Expectations**

🌒 Problem Solving Approach

UMPIRE: Understand, Match, Plan, Implement, Review, Evaluate.

We'll apply these six steps to the problems we'll see in the first half of the course.

We will learn to:

- **Understand** the problem
- **Match** identifies common approaches you've seen/used before
- **Plan** a solution step-by-step, and
- **Implement** the solution
- **Review** your solution
- **Evaluate** your solution's time and space complexity and think critically about the advantages and disadvantages of your chosen approach.

Breakout Problems Session 2

💡 Unit 10 Cheatsheet

This cheatsheet outlines content that will enhance and optimize your solutions to Unit 10 problems but *are not strictly necessary to solve the problems*.

⚠️ Approaching Unit 10 Problems

To help transition you from the CodePath classroom to an independent study environment post-classroom and to better prepare you for interviews, this unit does not contain hints.

Problems may cover any topic from Unit 1-9, and it is your job to match the problem to the most appropriate algorithmic technique and/or data structure. Some problems may not match any specific technique, but will require to use your general knowledge and the problem solving skills you have strengthened over the past nine units to find a solution.

Some problems may be repeated to you from past units. This is *intentional*. Use these problems as an opportunity to review techniques. For an extra challenge, use your additional knowledge to come to an even better solution than the one you developed earlier in the course. Check the cheatsheet for syntax and data structures that will help you write cleaner, more optimal code!

Happy coding!

Problem Set Version 1

Given an integer array `nums`, return `True` if any value appears **at least twice** in the array, and return `False` if every element is distinct.

```python
def contains_duplicate(nums):
	pass
```

Example Usage:

```
Example #1:
Input: nums = [1,2,3,1]
Output: True

Example #2:
Input: nums = [1,2,3,4]
Output: False

Example #3:
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: True
```

Problem 2: Remove Element

Given a list of integers `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. The order of the elements may be changed. Then return *the number of elements in* `nums` *which are not equal to* `val`.

Consider the number of elements in `nums` which are not equal to `val` be `k`, for your response to be acceptable, you need to do the following things:

- Change the list `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`

```python
def remove_element(nums, val):
	pass
```

```
Example #1:
Input: nums = [3,2,2,3], val = 3
Expected Output: 2
nums should be [2,2,_,_]
Explanation: Your function should return k = 2,
with the first two elements of nums being 2.
```

```
Input: nums = [0,1,2,2,3,0,4,2], val = 2
Output: 5
nums should be [0,1,4,0,3,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums containi
Note that the five elements can be returned in any order.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Problem 3: Greatest Common Divisor of Strings

For two strings `s` and `t`, we say "`t` divides `s`" if and only if `s = t + t + t + ... + t + t` (i.e., `t` is concatenated with itself one or more times).

Given two strings `str1` and `str2`, return *the largest string* `x` *such that* `x` *divides both* `str1` *and* `str2`.

```python
def gcd_of_stings(str1, str2):
	pass
```

```
Example #1:
Input: str1 = "ABCABC", str2 = "ABC"
Output: "ABC"

Example #2:
Input: str1 = "ABABAB", str2 = "ABAB"
Output: "AB"

Example #3:
Input: st1 = "LEET", str2 = "CODE"
Output: ""
```

Problem 4: Check Balanced Binary Tree

Given the `root` of a binary tree, return `True` if the tree is balanced and `False` otherwise.

A balanced binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

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

Example Usage:

```
Input Tree #1:

      3
     / \
    9   20
       /  \
      15   7

Input: root = 3
Output: True

Input Tree #2:

          1
         / \
        2   2
       / \
      3   3
     / \
    4   4

Input: root = 1
Output: False


Input Tree #3: Empty Tree
Input: root = 1
Output: True
```

Problem 5: Subarray Sum Equals K

Given an array of integers `nums` and an integer `k`, return *the total number of subarrays whose sum equals to* `k`.

A subarray is a contiguous **non-empty** sequence of elements within an array.

```python
def subarray_sum(nums, k):
        pass
```

Example Usage:

```
Output: 2

Example #2:
Input: nums = [1, 2, 3], k = 3
Output: 2
```

Problem 6: Add Two Numbers Represented By Linked Lists

You are given the heads of two **non-empty** linked lists `l1` and `l2` representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself

```python
class Node:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def add_two_numbers(l1, l2):
        pass
```

Example Usage:

```
Example Input Lists
list1: 2 -> 4 -> 3
list2: 5 -> 6 -> 4

Example Input: l1= 2, l2 = 5
Expected Output: 7
Expected Output List: 7 -> 0 -> 8
Explanation: 342 + 465 = 807
```

Close Section

▸ **Problem Set Version 2**

▸ **Problem Set Version 3**