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 1: Working with Linked Lists

In this session, students will learn to apply Python classes and linked lists through practical exercises. They will begin by creating and manipulating instances of a Pokémon class and then explore the basics of linked lists, focusing on node creation and linkage. These exercises aim to deepen understanding of object-oriented programming and provide foundational skills in managing custom data structures in Python.

> 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**

We will approach problems using the six steps in the UMPIRE 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 1

💡 Unit 6 Cheatsheet

To help your learning journey with linked lists, we've put together a guide to common concepts and syntax you will use throughout Unit 6 breakout problems. Use this cheatsheet as a quick reference guide as you work through the problems below.

▼ Problem Set Version 1

Problem 1: Nested Constructors

Step 1: Copy the following code into your IDE.

Step 2: Add a line of code (outside of the class) to create the linked list `4 -> 3 -> 2` in a single assignment statement.

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

▶ 💡 **Hint: Nested Constructors**

Problem 2: Find Frequency

Given the `head` of a linked list and a value `val`, return the frequency of `val` in the list. Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

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

def count_element(head, val):
        pass
```

Example Usage:

```
# Input List: 3 -> 1 -> 2 -> 1
# Input: head = 3, val = 1
```

Example Output:

```
# 2
```

Problem 3: Remove Tail

The following code attempts to remove the tail of a singly linked list. However, it has a bug!

Step 1: Copy this code into your IDE.

Step 2: Create your own test cases to run the code against, and use print statements and the stack trace to identify and fix the bug so that the function correctly removes the tail of the list.

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


# Helper function to print the linked list
def print_list(node):
    current = node
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()


# I have a bug!
```

```
    if head.next is None: # If there's only one node, removing it leaves the list empty
        return None

        # Start from the head and find the second-to-last node
    current = head
    while current.next:
        current = current.next

    current.next = None # Remove the last node by setting second-to-last node to None
    return head
```

Example Usage:

```
# Input List: 1 -> 2 -> 3 -> 4
# Input: head = 1
```

Example Output:

```
# Expected Return Value: 1
# Expected Result List: 1 -> 2 -> 3
```

Problem 4: Find the Middle

A variation of the two-pointer technique introduced in Unit 4 is to have a slow and a fast pointer that increment at different rates. Given the head of a linked list, use the slow-fast pointer technique to find the middle node of a linked list. If there are two middle nodes, return the second middle node.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

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

def find_middle_element(head):
        pass
```

Example Usage:

```
# Input List:
# 1 -> 2 -> 3
# Input: head = 1
```

Example Output:

```
# Expected Return Value: 2
```

Problem 5: Is Palindrome?

Given the head of a singly linked list, return `True` if the values of the linked list are palindromic and `False` otherwise. Use the two-pointer technique in your solution.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

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

def is_palindrome(head):
        pass
```

Example Usage:

```
# Input List:
# 1 -> 2 -> 1
# Input: head = 1
```

Example Output:

```
# True
```

▶ ✨ AI Hint: Multiple Pass Technique

Problem 6: Put it in Reverse

Given the `head` of a singly linked list, reverse the list, and return the reversed list. You must reverse the list in place. Return the head of the reversed list.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

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

def reverse(head):
        pass
```

```
# Input: head = 1
```

Example Output:

```
# Expected Return Value: 4
# Expected Result List: 4 -> 3 -> 2 -> 1
```

▶ 💡 **Hint: Which technique?**

Close Section

▶ Problem Set Version 2

▶ Problem Set Version 3