TIP102 | Intermediate Technical Interview Prep

Intermediate Technical Interview Prep Spring 2025 (@ Section 3 | Tuesdays and Thursdays 6PM - 8PM PT)
Personal Member ID#: 117667

Session 1: Strings & Arrays

Session Overview

Students will be introduced to the foundational concepts of strings and arrays in Python, which are essential for solving common coding problems. They will also learn about the UPI method, a structured approach to planning and solving technical interview questions. The lesson will cover basic operations like accessing, iterating over, and modifying lists, as well as performing advanced string manipulation.

> You can find all resources from today including session slide decks, session recordings, and more on the resources tab

📊 Part 1 : Instructor Led 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

To build a long-term organized approach to problem solving, we'll start with three main steps. We'll refer to them as **UPI: Understand, Plan, and Implement**.

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

We will learn to:

- **Understand** the problem,
- **Plan** a solution step-by-step, and
- **Implement** the solution

▶ **Comment on UPI**

▶ **UPI Example**

Breakout Problems Session 1

▶ Standard Problem Set Version 1

▶ Standard Problem Set Version 2

▼ Advanced Problem Set Version 1

Problem 1: Hunny Hunt

Write a function `linear_search()` to help Winnie the Pooh locate his lost items. The function accepts a list `items` and a `target` value as parameters. The function should return the first index of `target` in `items`, and `-1` if `target` is not in the `lst`. Do not use any built-in functions.

```
def linear_search(lst, target):
        pass
```

Example Usage:

```
items = ['haycorn', 'haycorn', 'haycorn', 'hunny', 'haycorn']
target = 'hunny'
linear_search(items, target)

items = ['bed', 'blue jacket', 'red shirt', 'hunny']
target = 'red balloon'
linear_search(items, target)
```

Example Output:

```
3
-1
```

▶ 💡 **Hint: Python Basics**

Problem 2: Bouncy, Flouncy, Trouncy, Pouncy

Tigger has developed a new programming language Tiger with only **four** operations and **one** variable `tigger`.

- `bouncy` and `flouncy` both **increment** the value of the variable `tigger` by `1`.
- `trouncy` and `pouncy` both **decrement** the value of the variable `tigger` by `1`.

Initially, the value of `tigger` is `1` because he's the only tigger around! Given a list of strings `operations` containing a list of operations, return the **final** value of `tigger` after performing all the operations.

```
def final_value_after_operations(operations):
        pass
```

Example Usage:

```
operations = ["trouncy", "flouncy", "flouncy"]
final_value_after_operations(operations)

operations = ["bouncy", "bouncy", "flouncy"]
final_value_after_operations(operations)
```

Example Output:

```
2
4
```

Problem 3: T-I-Double Guh-Er II

T-I-Double Guh-Er: That spells Tigger! Write a function `tiggerfy()` that accepts a string `word` and returns a new string that removes any substrings `t`, `i`, `gg`, and `er` from `word`. The function should be case insensitive.

```
def tiggerfy(word):
        pass
```

Example Usage:

```
word = "Trigger"
tiggerfy(word)

word = "eggplant"
tiggerfy(word)

word = "Choir"
tiggerfy(word)
```

Example Output:

```
"r"
"eplan"
"Choir"
```

▶ 💡 **Hint: String Methods**

Problem 4: Non-decreasing Array

Given an array `nums` with `n` integers, write a function `non_decreasing()` that checks if `nums` could become non-decreasing by modifying **at most one element**.

We define an array is non-decreasing if `nums[i] <= nums[i + 1]` holds for every `i` (**0-based**) such that (`0 <= i <= n - 2`).

```
def non_decreasing(nums):
        pass
```

Example Usage:

```
nums = [4, 2, 3]
non_decreasing(nums)

nums = [4, 2, 1]
non_decreasing(nums)
```

Example Output:

```
True
False
```

Problem 5: Missing Clues

Christopher Robin set up a scavenger hunt for Pooh, but it's a blustery day and several hidden clues have blown away. Write a function `find_missing_clues()` to help Christopher Robin figure out which clues he needs to remake. The function accepts two integers `lower` and `upper` and a unique integer array `clues`. All elements in `clues` are within the inclusive range `[lower, upper]`.

A clue `x` is considered missing if `x` is in the range `[lower, upper]` and `x` is not in `clues`.

Return the shortest sorted list of ranges that exactly covers all the missing numbers. That is, no element of `clues` is included in any of the ranges, and each missing number is covered by one of the ranges.

```
def find_missing_clues(clues, lower, upper):
   pass
```

Example Usage:

```
clues = [0, 1, 3, 50, 75]
lower = 0
upper = 99
find_missing_clues(clues, lower, upper)

clues = [-1]
lower = -1
upper = -1
find_missing_clues(clues, lower, upper)
```

Example Output:

```
[[2, 2], [4, 49], [51, 74], [76, 99]]
[]
```

▶ 💡 **Hint: Nested Lists**

▶ 💡 **Hint: String Methods**

Problem 6: Vegetable Harvest

Rabbit is collecting carrots from his garden to make a feast for Pooh and friends. Write a function `harvest()` that accepts a 2D `n x m` matrix `vegetable_patch` and returns the number of of carrots that are ready to harvest in the vegetable patch. A carrot is ready to harvest if `vegetable_patch[i][j]` has value `'c'`.

Assume `n = len(vegetable_patch)` and `m = len(vegetable_patch[0])`. `0 <= i < n` and `0 <= j < m`.

```
def harvest(vegetable_patch):
        pass
```

Example Usage:

```
vegetable_patch = [
        ['x', 'c', 'x'],
        ['x', 'x', 'x'],
        ['x', 'c', 'c'],
        ['c', 'c', 'c']
]
harvest(vegetable_patch)
```

Example Output:

```
6
```

▶ 💡 **Hint: Nested Loops**

Problem 7: Eeyore's House

Eeyore has collected two piles of sticks to rebuild his house and needs to choose pairs of sticks whose lengths are the right proportion. Write a function `good_pairs()` that accepts two integer arrays `pile1` and `pile2` where each integer represents the length of a stick. The function also accepts a positive integer `k`. The function should return the number of **good** pairs.

A pair `(i, j)` is called **good** if `pile1[i]` is divisible by `pile2[j] * k`. Assume `0 <= i <= len(pile1) - 1` and `0 <= j <= len(pile2) - 1`.

```
def good_pairs(pile1, pile2, k):
        pass
```

Example Usage:

```
pile1 = [1, 3, 4]
pile2 = [1, 3, 4]
k = 1
good_pairs(pile1, pile2, k)

pile1 = [1, 2, 4, 12]
pile2 = [2, 4]
k = 3
good_pairs(pile1, pile2, k)
```

Example Output:

```
5
2
```

▶ 💡 **Remainders with Modulus Division**

Problem 8: Local Maximums

Write a function `local_maximums()` that accepts an `n x n` integer matrix `grid` and returns an integer matrix `local_maxes` of size `(n - 2) x (n - 2)` such that:

- `local_maxes[i][j]` is equal to the largest value of the `3 x 3` matrix in `grid` centered around row `i + 1` and column `j + 1`.

In other words, we want to find the largest value in every contiguous `3 x 3` matrix in `grid`.

```
def local_maximums(grid):
        pass
```

9	9	8	1
5	6	2	6
8	2	6	4
6	2	2	2

9	9
8	6

Example Usage:

```
grid = [
        [9, 9, 8, 1],
        [5, 6, 2, 6],
        [8, 2, 6, 4],
        [6, 2, 2, 2]
]
local_maximums(grid)

grid = [
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 2, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]
]
local_maximums(grid)
```

Example Output:

```
[[9, 9], [8, 6]]
[[2, 2, 2], [2, 2, 2], [2, 2, 2]]
```

Close Section

▾ Advanced Problem Set Version 2

Problem 1: Words Containing Character

Write a function `words_with_char()` that accepts a list of strings `words` and a character `x`. Return a list of indices representing the words that contain the character `x`. The returned list may be in any order.

```
def words_with_char(words, x):
        pass
```

Example Usage:

```
words = ["batman", "superman"]
x = "a"
words_with_char(words, x)

words = ["black panther", "hulk", "black widow", "thor"]
x = "a"
words_with_char(words, x)

words = ["star-lord", "gamora", "groot", "rocket"]
x = "z"
words_with_char(words, x)
```

Example Output:

```
[0, 1]
[0, 2]
[]
```

▶ 💡 **Hint: Python Basics**

Problem 2: HulkSmash

Write a function `hulk_smash()` that accepts an integer `n` and returns a 1-indexed string array `answer` where:

- `answer[i] == "HulkSmash"` if `i` is divisible by `3` and `5`.
- `answer[i] == "Hulk"` if `i` is divisible by `3`.
- `answer[i] == "Smash"` if `i` is divisible by `5`.
- `answer[i] == i` (as a string) if none of the above conditions are true.

```
def hulk_smash(n):
        pass
```

Example Usage:

```
n = 3
hulk_smash(n)

n = 5
hulk_smash(n)

n = 15
hulk_smash(n)
```

Example Output:

```
["1", "2", "Hulk"]
["1", "2", "Hulk", "4", "Smash"]
["1", "2", "Hulk", "4", "Smash", "Hulk", "7", "8", "Hulk", "Smash", "11", "Hulk", "1
```

Problem 3: Encode

The Riddler is planning to leave a coded message to lead Batman into a trap. Write a function `shuffle()` that takes in a string, the Riddler's `message`, and encodes it using an integer array `indices`. The message will be shuffled such that the character at the `ith` position in `message` moves to index `indices[i]` in the shuffled string. You may assume `len(message)` is equal to the `len(indices)`.

```
def shuffle(message, indices):
        pass
```

Example Usage:

```
message = "evil"
indices = [3, 1, 2, 0]
shuffle(message, indices)

message = "findme"
indices = [0, 1, 2, 3, 4, 5]
shuffle(message, indices)
```

Example Output:

```
"lvie"
"findme"
```

▶ 💡 **Hint: String Methods**

Problem 4: Good Samaritan

Superman is doing yet another good deed, using his power of flight to distribute meals for the Metropolis Food Bank. He wants to distribute meals in the least number of trips possible.

Metropolis Food Bank currently stores meals in `n` packs where the `ith` pack contains `meals[i]` meals. There are also `m` empty boxes which can contain up to `capacity[i]` meals.

Given an array `meals` of length `n` and `capacity` of size `m`, write a function `minimum_boxes()` that returns the **minimum** number of boxes needed to redistribute the `n` packs of meals into boxes.

Note that meals from the same pack can be distributed into different boxes.

```
def minimum_boxes(meals, capacity):
        pass
```

Example Usage:

```
meals = [1, 3, 2]
capacity = [4, 3, 1, 5, 2]
minimum_boxes(meals, capacity)

meals = [5, 5, 5]
capacity = [2, 4, 2, 7]
minimum_boxes(meals, capacity)
```

Example Output:

```
2
4
```

▶ 💡 **Hint: Sorting Lists**

Problem 5: Heist

The legendary outlaw Robin Hood is looking for the target of his next heist. Write a function `wealthiest_customer()` that accepts an `m x n` 2D integer matrix `accounts` where `accounts[i][j]` is the amount of money the `ith` customer has in the `jth` bank. Return a list `[i, w]` where `i` is the 0-based index of the wealthiest customer and `w` is the total wealth of the wealthiest customer.

If multiple customers have the highest wealth, return the index of any customer.

A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth.

```
def wealthiest_customer(accounts):
        pass
```

Example Usage:

```
accounts = [
        [1, 2, 3],
        [3, 2, 1]
]
wealthiest_customer(accounts)

accounts = [
        [1, 5],
        [7, 3],
        [3, 5]
]
wealthiest_customer(accounts)

accounts = [
        [2, 8, 7],
        [7, 1, 3],
        [1, 9, 5]
]
wealthiest_customer(accounts)
```

Example Output:

```
[0, 6]
[1, 10]
[0, 17]
```

▶ 💡 **Hint: Nested Lists**

Problem 6: Smaller Than

Write a function `smaller_than_current` that accepts a list of integers `nums` and, for each element in the list `nums[i]`, determines the number of other elements in the array that are smaller than it. More formally, for each `nums[i]` count the number of valid `j`'s such that `j != i` and `nums[j] < nums[i]`.

Return the answer as a list.

```
def smaller_than_current(nums):
        pass
```

Example Usage:

```
nums = [8, 1, 2, 2, 3]
smaller_than_current(nums)

nums = [6, 5, 4, 8]
smaller_than_current(nums)

nums = [7, 7, 7, 7]
smaller_than_current(nums)
```

Example Output:

```
[4, 0, 1, 1, 3]
[2, 1, 0, 3]
[0, 0, 0, 0]
```

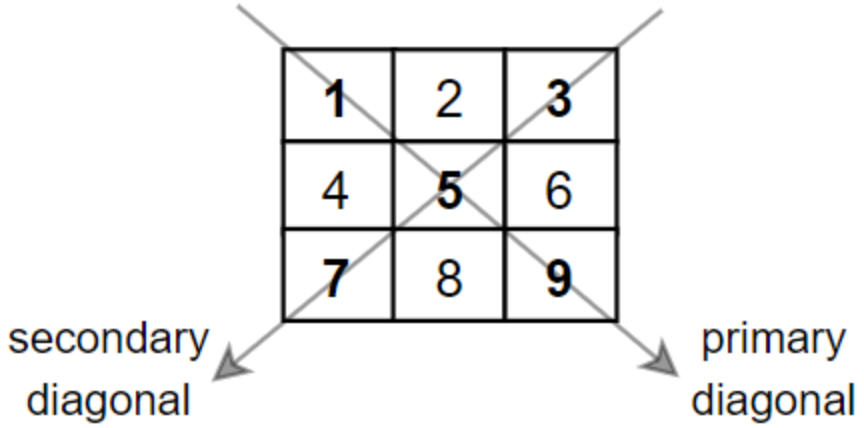
▶ 💡 **Hint: Nested Loops**

Problem 7: Diagonal

Write a function `diagonal_sum()` that accepts a 2D `n x n` matrix `grid` and returns the sum of the matrix diagonals. Only include the sum of all the elements on the primary diagonal and all the elements in the secondary diagonal that are not part of the primary diagonal.

The primary diagonal is all cells in the matrix along a line drawn from the top-left cell in the matrix to the bottom-right cell. The secondary diagonal is all cells in the matrix along a line drawn from the top-right cell in the matrix to the bottom-left cell.

```
def diagonal_sum(grid):
        pass
```



Example Usage

```
grid = [
        [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
diagonal_sum(grid)

grid = [
        [1, 1, 1, 1],
    [1, 1, 1, 1],
        [1, 1, 1, 1],
    [1, 1, 1, 1]
]
diagonal_sum(grid)

grid = [
        [5]
]
diagonal_sum(grid)
```

Example Output:

```
25
8
5
```

Problem 8: Defuse the Bomb

Batman has a bomb to defuse, and his time is running out! His butler, Alfred, is on the phone providing him with a circular array `code` of length `n` and key `k`.

To decrypt the code, Batman must replace every number. All the numbers are replaced simultaneously.

- If `k > 0`, replace the ith number with the sum of the next k numbers.
- If `k < 0`, replace the ith number with the sum of the previous k numbers.

- If `k == 0`, replace the ith number with 0.

As `code` is circular, the next element of `code[n-1]` is `code[0]`, and the previous element of `code[0]` is `code[n-1]`.

Given the circular array `code` and an integer key `k`, write a function `decrypt()` that returns the decrypted code to defuse the bomb!

```
def defuse(code, k):
        pass
```

Example Usage:

```
code = [5, 7, 1, 4]
k = 3
defuse(code, k)

code = [1, 2, 3, 4]
k = 0
defuse(code, k)

code = [2, 4, 9, 3]
k = -2
defuse(code, k)
```

Example Output:

```
[12, 10, 16, 13]
[0, 0, 0, 0]
[12, 5, 6, 13]
```

▶ 💡 **Remainders with Modulus Division**

Close Section