TIP102 | Intermediate Technical Interview Prep

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

Session 2: Strings & Arrays

Session Overview

Students will build upon the concepts introduced in the first session by tackling more complex string and array problems. They will deepen their understanding of manipulating data structures, including reversing words in a sentence, summing digits, and handling list operations such as finding exclusive elements between two lists. The session will also introduce them to important problem-solving techniques, like using loops and conditionals to manipulate numbers and strings.

> 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 2

▼ Standard Problem Set Version 1

Problem 1: Reverse Sentence

Write a function `reverse_sentence()` that takes in a string `sentence` and returns the sentence with the order of the words reversed. The sentence will contain only alphabetic characters and spaces to separate the words. If there is only one word in the sentence, the function should return the original string.

```
def reverse_sentence(sentence):
    pass
```

Example Usage:

```
sentence = "tubby little cubby all stuffed with fluff"
reverse_sentence(sentence)

sentence = "Pooh"
reverse_sentence(sentence)
```

Example Output:

```
"fluff with stuffed all cubby little tubby"
"Pooh"
```

▶ 💡 **Hint: String Methods**

Problem 2: Goldilocks Number

In the extended universe of fictional bears, Goldilocks finds an enticing list of numbers in the Three Bears' house. She doesn't want to take a number that's too high or too low - she wants a number that's juuust right. Write a function `goldilocks_approved()` that takes in the list of distinct positive integers `nums` and returns any number from the list that is neither the minimum nor the maximum value in the array, or `-1` if there is no such number.

Return the selected integer.

```
def goldilocks_approved(nums):
    pass
```

Example Usage

```
nums = [3, 2, 1, 4]
goldilocks_approved(nums)

nums = [1, 2]
goldilocks_approved(nums)

nums = [2, 1, 3]
goldilocks_approved(nums)
```

Example Output:

```
2
-1
2
```

▶ 💡 **Hint: Minimums and Maximums**

Problem 3: Delete Minimum

Pooh is eating all of his hunny jars in order of smallest to largest. Given a list of integers `hunny_jar_sizes`, write a function `delete_minimum_elements()` that continuously removes the minimum element until the list is empty. Return a new list of the elements of `hunny_jar_sizes` in the order in which they were removed.

```
def delete_minimum_elements(hunny_jar_sizes):
        pass
```

Example Usage

```
hunny_jar_sizes = [5, 3, 2, 4, 1]
delete_minimum_elements(hunny_jar_sizes)

hunny_jar_sizes = [5, 2, 1, 8, 2]
delete_minimum_elements(hunny_jar_sizes)
```

Example Output:

```
[1, 2, 3, 4, 5]
[1, 2, 2, 5, 8]
```

▶ 💡 **Hint: While Loops**

Problem 4: Sum of Digits

Write a function `sum_of_digits()` that accepts an integer `num` and returns the sum of `num`'s digits.

```
def sum_of_digits(num):
    pass
```

Example Usage

```
num = 423
sum_of_digits(num)

num = 4
sum_of_digits(num)
```

Example Output:

```
9 # Explanation: 4 + 2 + 3 = 9
4
```

▶ 💡 **Hint: Floor Division**

Problem 5: 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 6: Acronym

Given an array of strings `words` and a string `s`, implement a function `is_acronym()` that returns `True` if `s` is an acronym of `words` and returns `False` otherwise.

The string `s` is considered an acronym of `words` if it can be formed by concatenating the first character of each string in `words` in order. For example, `"pb"` can be formed from `["pooh"", "bear"]`, but it can't be formed from `["bear", "pooh"]`.

```
def is_acronym(words, s):
    pass
```

Example Usage

```
words = ["christopher", "robin", "milne"]
s = "crm"
is_acronym(words, s)
```

Example Output:

```
True
```

Problem 7: Good Things Come in Threes

Write a function `make_divisible_by_3()` that accepts an integer array `nums`. In one operation, you can add or subtract `1` from any element of `nums`. Return the minimum number of operations to make all elements of nums divisible by 3.

```
def make_divisible_by_3(nums):
    pass
```

Example Usage

```
nums = [1, 2, 3, 4]
make_divisible_by_3(nums)

nums = [3, 6, 9]
make_divisible_by_3(nums)
```

Example Output:

```
3
0
```

▶ 💡 **Remainders with Modulus Division**

Problem 8: Exclusive Elements

Given two lists `lst1` and `lst2`, write a function `exclusive_elemts()` that returns a new list that contains the elements which are in `lst1` but not in `lst2` and the elements that are in `lst2` but not in `lst1`.

```
def exclusive_elemts(lst1, lst2):
        pass
```

Example Usage

```
lst1 = ["pooh", "roo", "piglet"]
lst2 = ["piglet", "eeyore", "owl"]
exclusive_elemts(lst1, lst2)

lst1 = ["pooh", "roo"]
lst2 = ["piglet", "eeyore", "owl", "kanga"]
exclusive_elemts(lst1, lst2)

lst1 = ["pooh", "roo", "piglet"]
lst2 = ["pooh", "roo", "piglet"]
exclusive_elemts(lst1, lst2)
```

Example Output:

```
["pooh", "roo", "eeyore", "owl"]
["pooh", "roo", "piglet", "eeyore", "owl", "kanga"]
[]
```

Problem 9: Merge Strings Alternately

Write a function `merge_alternately()` that accepts two strings `word1` and `word2`. Merge the strings by adding letters in alternating order, starting with `word1`. If a string is longer than the other, append the additional letters onto the end of the merged string.

Return the merged string.

```
def merge_alternately(word1, word2):
    pass
```

Example Usage

```
word1 = "wol"
word2 = "oze"
merge_alternately(word1, word2)

word1 = "hfa"
word2 = "eflump"
merge_alternately(word1, word2)

word1 = "eyre"
word2 = "eo"
merge_alternately(word1, word2)
```

Example Output:

```
"woozle"
"heffalump"
"eeyore"
```

Problem 10: 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
```

▶ 💡 **Hint: Nested Loops**

Close Section

▼ Standard Problem Set Version 2

Problem 1: String Array Equivalency

Given two string arrays `word1` and `word2`, return `True` if the two arrays **represent** the same string, and `False` otherwise.

A string is **represented** by an array if the array elements concatenated in order forms the string.

```
def are_equivalent(word1, word2):
    pass
```

Example Usage

```
word1 = ["bat", "man"]
word2 = ["b", "atman"]
are_equivalent(word1, word2)

word1 = ["alfred", "pennyworth"]
word2 = ["alfredpenny", "word"]
are_equivalent(word1, word2)

word1  = ["cat", "wom", "an"]
word2 = ["catwoman"]
are_equivalent(word1, word2)
```

Example Output:

```
True
False
True
```

▶ 💡 **Hint: String Methods**

Problem 2: Count Even Strings

Implement a function `count_evens()` that accepts a list of strings `lst` as a parameter. The function should return the number of strings with an even length in the list.

```
def count_evens(lst):
    pass
```

Example Usage

```
lst = ["na", "nana", "nanana", "batman", "!"]
count_evens(lst)

lst = ["the", "joker", "robin"]
count_evens(lst)

lst = ["you", "either", "die", "a", "hero", "or", "you", "live", "long", "enough", "
count_evens(lst)
```

Example Output:

```
4
0
9
```

▶ 💡 **Remainders with Modulus Division**

Problem 3: Secret Identity

Write a function `remove_name()` to keep Batman's secret identity hidden. The function accepts a list of names `people` and a string `secret_identity` and should return the list with any instances of `secret_identity` removed. The list must be modified in place; you may not create any new lists as part of your solution. Relative order of the remaining elements must be maintained.

```
def remove_name(people, secret_identity):
        pass
```

Example Usage

```
people = ['Batman', 'Superman', 'Bruce Wayne', 'The Riddler', 'Bruce Wayne']
secret_identity = 'Bruce Wayne'
remove_name(people, secret_identity)
```

Example Output:

```
['Batman', 'Superman', 'The Riddler']
```

▶ 💡 **Hint: While Loops**

Problem 4: Count Digits

Given a non-negative integer `n`, write a function `count_digits()` that returns the number of digits in `n`. You may not cast `n` to a string.

```
def count_digits(n):
    pass
```

Example Usage

```
n = 964
count_digits(n)

n = 0
count_digits(n)
```

Example Output:

```
3
1
```

▶ 💡 **Hint: Floor Division**

Problem 5: Move Zeroes

Write a function `move_zeroes` that accepts an integer array `nums` and returns a new list with all `0`s moved to the end of list. The relative order of the non-zero elements in the original list should be maintained.

```
def move_zeroes(lst):
    pass
```

Example Usage

```
lst = [1, 0, 2, 0, 3, 0]
move_zeroes(lst)
```

Example Output:

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

Problem 6: Reverse Vowels of a String

Given a string `s`, reverse only all the vowels in the string and return it.

The vowels are `'a'`, `'e'`, `'i'`, `'o'`, and `'u'`, and they can appear in both lower and upper cases and more than once.

```
def reverse_vowels(s):
    pass
```

Example Usage

```
s = "robin"
reverse_vowels(s)

s = "BATgirl"
reverse_vowels(s)

s = "batman"
reverse_vowels(s)
```

Example Output:

```
"ribon"
"BiTgArl"
"batman"
```

Problem 7: Vantage Point

Batman is going on a scouting trip, surveying an area where he thinks Harley Quinn might commit her next crime spree. The area has many hills with different heights and Batman wants to find the tallest one to get the best vantage point. His scout trip consists of `n + 1` points at different altitudes. Batman starts his trip at point `0` with altitude `0`.

Write a function `highest_altitude()` that accepts an integer array `gain` of length `n` where `gain[i]` is the net gain in altitude between points `i` and `i + 1` for all `(0 <= i < n)`. Return the highest altitude of a point.

```
def highest_altitude(gain):
    pass
```

Example Usage

```
gain = [-5, 1, 5, 0, -7]
highest_altitude(gain)

gain = [-4, -3, -2, -1, 4, 3, 2]
highest_altitude(gain)
```

Example Output:

```
1
0
```

Problem 8: Left and Right Sum Differences

Given a 0-indexed integer array `nums`, write a function `left_right_difference` that returns a 0-indexed integer array `answer` where:

- `len(answer) == len(nums)`
- `answer[i]` = `left_sum[i] - right_sum[i]`

Where:

- `left_sum[i]` is the sum of elements to the left of the index `i` in the array `nums`. If there is no such element, `left_sum[i] = 0`
- `right_sum[i]` is the sum of elements to the right of the index `i` in the array `nums`. If there is no such element, `right_sum[i] = 0`

```
def left_right_difference(nums):
    pass
```

Example Usage

```
nums = [10, 4, 8, 3]
left_right_difference(nums)

nums = [1]
left_right_difference(nums)
```

Example Output:

```
[-15, -1, 11, 22]
[0]
```

Problem 9: Common Cause

Write a function `common_elements()` that takes in two lists `lst1` and `lst2` and returns a list of the elements that are common to both lists.

```
def common_elements(lst1, lst2):
    pass
```

```
Example 1:
Input: lst1 = ["super strength", "super speed", "x-ray vision"], lst2 = ["super spee
Output: ["super speed"]

Example 2:
Input: lst1 = ["super strength", "super speed", "x-ray vision"], lst2 = ["martial ar
Output: []
```

Example Usage

```
lst1 = ["super strength", "super speed", "x-ray vision"]
lst2 = ["super speed", "time travel", "dimensional travel"]
common_elements(lst1, lst2)

lst1 = ["super strength", "super speed", "x-ray vision"]
lst2 = ["martial arts", "stealth", "master detective"]
common_elements(lst1, lst2)
```

Example Output:

```
["super speed"]
[]
```

▶ 💡 **Hint: Nested Loops**

Problem 10: Exposing Superman

Metropolis has a population `n`, with each citizen assigned an integer id from `1` to `n`. There's a rumor that Superman is an ordinary citizen among this group.

If Superman is an ordinary citizen, then:

- Superman trusts nobody.
- Everybody (except for Superman) trusts Superman.
- There is exactly one citizen that satisfies properties 1 and 2.

Write a function `expose_superman()` that accepts a 2D array `trust` where `trust[i] = [ai, bi]` representing that the person labeled `ai` trusts the person labeled `bi`. If a trust relationship does not exist in `trust` array, then such a trust relationship does not exist.

Return the label of Superman if he is hiding amongst the population and can be identified, or return `-1` otherwise.

```
def expose_superman(trust, n):
    pass
```

Example Usage

```
n = 2
trust = [[1, 2]]
expose_superman(trust, n)

n = 3
trust = [[1, 3], [2, 3]]
expose_superman(trust, n)

n = 3
trust = [[1, 3], [2, 3], [3, 1]]
expose_superman(trust, n)
```

Example Output:

```
2
3
-1
```

Close Section

▸ Advanced Problem Set Version 1

▸ Advanced Problem Set Version 2