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: Dictionaries & Sets

Session Overview

Students will continue to expand their expertise in Python through the exploration of data structures such as lists, dictionaries, and sets. They engage in various tasks like verifying list properties, creating and updating dictionaries, and analyzing data to make decisions.

> 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: Most Endangered Species

Imagine you are working on a wildlife conservation database. Write a function named `most_endangered()` that returns the species with the highest conservation priority based on its population.

The function should take in a list of dictionaries named `species_list` as a parameter. Each dictionary represents data associated with a species, including its `name`, `habitat`, and wild `population`. The function should return the `name` of the species with the lowest `population`.

If there are multiple species with the lowest population, return the species with the *lowest* index.

```
def most_endangered(species_list):
    pass
```

Example Usage:

```
species_list = [
    {"name": "Amur Leopard",
     "habitat": "Temperate forests",
     "population": 84
    },
    {"name": "Javan Rhino",
     "habitat": "Tropical forests",
     "population": 72
    },
    {"name": "Vaquita",
     "habitat": "Marine",
     "population": 10
    }
]

print(most_endangered(species_list))
```

Example Output:

```
Vaquita
```

▶ 💡 **Hint: Accessing Values in a Dictionary**

▶ 💡 **Hint: Accessing Keys, Values, and Key-Value Pairs**

Problem 2: Identifying Endangered Species

As part of conservation efforts, certain species are considered endangered and are represented by the string `endangered_species`. Each character in this string denotes a different endangered species. You also have a record of all observed species in a particular region, represented by the string `observed_species`. Each character in `observed_species` denotes a species observed in the region.

Your task is to determine how many instances of the observed species are also considered endangered.

Note: Species are case-sensitive, so "a" is considered a different species from "A".

Write a function to count the number of endangered species observed.

```
def count_endangered_species(endangered_species, observed_species):
    pass
```

Example Usage:

```
endangered_species1 = "aA"
observed_species1 = "aAAbbbb"

endangered_species2 = "z"
observed_species2 = "ZZ"

print(count_endangered_species(endangered_species1, observed_species1))
print(count_endangered_species(endangered_species2, observed_species2))
```

Example Output:

```
3 # `a` and `A` are endangered species. `a` appears once, and `A` twice.
0
```

▶ 💡 **Hint: Introduction to sets**

Problem 3: Navigating the Research Station

In a wildlife research station, each letter of the alphabet represents a different observation point laid out in a single row. Given a string `station_layout` of length `26` indicating the layout of these observation points (indexed from `0` to `25`), you start your journey at the first observation point (index `0`). To make observations in a specific order represented by a string `observations`, you need to move from one point to another.

The time taken to move from one observation point to another is the absolute difference between their indices, `|i - j|`.

Write a function that returns the total time it takes to visit all the required observation points in the given order with one movement.

```
def navigate_research_station(station_layout, observations):
  pass
```

Example Usage:

```
station_layout1 = "pqrstuvwxyzabcdefghijklmno"
observations1 = "wildlife"

station_layout2 = "abcdefghijklmnopqrstuvwxyz"
observations2 = "cba"

print(navigate_research_station(station_layout1, observations1))
print(navigate_research_station(station_layout2, observations2))
```

Example Output:

```
45
4
Example 2 explanation: The index moves from 0 to 2 to observe 'c', then to 1 for
'b', then to 0 again for 'a'.
Total time = 2 + 1 + 1 = 4.
```

▶ 💡 **Hint: What should my keys and values be?**

▶ 💡 **Hint: `enumerate()` Function**

Problem 4: Prioritizing Endangered Species Observations

In your work with a wildlife conservation database, you have two lists: `observed_species` and `priority_species`. The elements of `priority_species` are distinct, and all elements in `priority_species` are also in `observed_species`.

Write a function `prioritize_observations()` that sorts the elements of `observed_species` such that the relative ordering of items in `observed_species` matches that of `priority_species`. Species that do not appear in `priority_species` should be placed at the end of `observed_species` in ascending order.

```
def prioritize_observations(observed_species, priority_species):
  pass
```

Example Usage:

```
observed_species1 = ["🐯", "🦁", "🦌", "🦁", "🐯", "🐘", "🐍", "🦑", "🐻", "🐯", "🐼
priority_species1 = ["🐯", "🦌", "🐘", "🦁"]

observed_species2 = ["bluejay", "sparrow", "cardinal", "robin", "crow"]
priority_species2 = ["cardinal", "sparrow", "bluejay"]

print(prioritize_observations(observed_species1, priority_species1))
print(prioritize_observations(observed_species2, priority_species2))
```

Expected Output:

```
["🐯", "🐯", "🐯", "🦌", "🐘", "🦁", "🦁", "🐻", "🦑", "🐼", "🐍"]
["cardinal", "sparrow", "bluejay", "crow", "robin"]
```

▶ 💡 **Hint: `extend()` Function**

Problem 5: Calculating Conservation Statistics

You are given a 0-indexed integer array `species_populations` of even length, where each element represents the population of a particular species in a wildlife reserve.

As long as `species_populations` is not empty, you must repetitively:

1. Find the species with the minimum population and remove it.
2. Find the species with the maximum population and remove it.
3. Calculate the average population of the two removed species.

The average of two numbers `a` and `b` is `(a+b)/2`.

For example, the average of `200` and `300` is `(200+300)/2=250`.

Return the number of distinct averages calculated using the above process.

Note that when there is a tie for a minimum or maximum population, any can be removed.

```python
def distinct_averages(species_populations):
  pass
```

Example Usage:

```python
species_populations1 = [4,1,4,0,3,5]
species_populations2 = [1,100]

print(distinct_averages(species_populations1))
print(distinct_averages(species_populations2))
```

Example Output:

```
2
Example 1 Explanation:
1. Remove 0 and 5, and the average is (0 + 5) / 2 = 2.5. Now, nums = [4,1,4,3].
2. Remove 1 and 4. The average is (1 + 4) / 2 = 2.5, and nums = [4,3].
3. Remove 3 and 4, and the average is (3 + 4) / 2 = 3.5.
Since there are 2 distinct numbers among 2.5, 2.5, and 3.5, we return 2.

1
Example 2 Explanation:
There is only one average to be calculated after removing 1 and 100,
so we return 1.
```

Problem 6: Wildlife Reintroduction

As a conservationist, your research center has been raising multiple endangered species and is now ready to reintroduce them into their native habitats. You are given two 0-indexed strings `raised_species` and `target_species`. The string `raised_species` represents the list of species available to release into the wild at your center, where each character represents a different species. The string `target_species` represents a specific sequence of species you want to form and release together.

You can take some species from `raised_species` and rearrange them to form new sequences.

Return the maximum number of copies of `target_species` that can be formed by taking species from `raised_species` and rearranging them.

```python
def max_species_copies(raised_species, target_species):
  pass
```

Example Usage:

```python


raised_species1 = "abcba"
target_species1 = "abc"
print(max_species_copies(raised_species1, target_species1))  # Output: 1

raised_species2 = "aaaaabbbbcc"
target_species2 = "abc"
print(max_species_copies(raised_species2, target_species2)) # Output: 2
```

Example Output:

```
1
Example 1 Explanation:
We can make one copy of "abc" by taking the letters at indices 0, 1, and 2.
We can make at most one copy of "abc", so we return 1.
Note that while there is an extra 'a' and 'b' at indices 3 and 4, we cannot
reuse the letter 'c' at index 2, so we cannot make a second copy of "abc".

2
Example 2 Explanation:
We can make one copy of "abc" by taking the letters at indices 0, 5, and 9.
We can make a second copy of "abc" by taking the letters at indices 1, 6, and 10
At this point we are out of the letter "c" and cannot make additional copies.
```

Problem 7: Count Unique Species

You are given a string `ecosystem_data` that consists of digits and lowercase English letters. The digits represent the observed counts of various species in a protected ecosystem.

You will replace every non-digit character with a space. For example, `"f123de34g8hi34"` will become `" 123 34 8 34"`. Notice that you are left with some species counts that are separated by at least one space: `"123", "34", "8", and "34"`.

Return the number of unique species counts after performing the replacement operations on `ecosystem_data`.

Two species counts are considered different if their decimal representations without any leading zeros are different.

```
def count_unique_species(ecosystem_data):
  pass
```

Example Usage:

```
ecosystem_data1 = "f123de34g8hi34"
ecosystem_data2 = "species1234forest234"
ecosystem_data3 = "x1y01z001"

print(count_unique_species(ecosystem_data1))
print(count_unique_species(ecosystem_data2))
print(count_unique_species(ecosystem_data3))
```

Example Output:

```
3
2
1
```

▶ 💡 **Hint: Representing Infinite Values**

Problem 8: Equivalent Species Pairs

In an effort to understand species diversity in different habitats, researchers are analyzing species pairs observed in various regions. Each pair is represented by a list `[a, b]` where `a` and `b` represent two species observed together.

A species pair `[a, b]` is considered equivalent to another pair `[c, d]` if and only if either `(a == c and b == d)` or `(a == d and b == c)`. This means that the order of species in a pair does not matter.

Your task is to determine the number of equivalent species pairs in the list of observed species pairs.

```
def num_equiv_species_pairs(species_pairs):
  pass
```

Example Usage:

```
species_pairs1 = [[1,2],[2,1],[3,4],[5,6]]
species_pairs2 = [[1,2],[1,2],[1,1],[1,2],[2,2]]

print(num_equiv_species_pairs(species_pairs1))
print(num_equiv_species_pairs(species_pairs2))
```

Example Output:

```
1
3
```

▶ 💡 **Hint: Calculating the Number of Equivalent Pairs**

Close Section

▾ Standard Problem Set Version 2

Problem 1: Filter Destinations

You're planning an epic trip and have a dictionary of destinations mapped to their respective rating scores. Your goal is to visit only the best-rated destinations. Write a function that takes in a dictionary `destinations` and a `rating_threshold` as parameters. The function should iterate through the dictionary and remove all destinations that have a rating strictly below the `rating_threshold`. Return the updated dictionary.

```python
def remove_low_rated_destinations(destinations, rating_threshold):
  pass
```

Example Usage:

```python
destinations = {"Paris": 4.8, "Berlin": 3.5, "Addis Ababa": 4.9, "Moscow": 2.8}
destinations2 = {"Bogotá": 4.8, "Kansas City": 3.9, "Tokyo": 4.5, "Sydney": 3.0}

print(remove_low_rated_destinations(destinations, 4.0))
print(remove_low_rated_destinations(destinations2, 4.9))
```

Example Output:

```
{"Paris": 4.8, "Addis Ababa": 4.9}
{}
```

▶ 💡 **Hint: Accessing Values in a Dictionary**

▶ 💡 **Hint: Accessing Keys, Values, and Key-Value Pairs**

Problem 2: Unique Travel Souvenirs

As a seasoned traveler, you've collected a variety of souvenirs from different destinations. You have an array of string `souvenirs`, where each string represents a type of souvenir. You want to know if the number of occurrences of each type of souvenir in your collection is unique.

Write a function that takes in an array `souvenirs` and returns `True` if the number of occurrences of each value in the array is unique, or `False` otherwise.

```
def unique_souvenir_counts(souvenirs):
  pass
```

Example Usage:

```
souvenirs1 = ["keychain", "hat", "hat", "keychain", "keychain", "postcard"]
souvenirs2 = ["postcard", "postcard", "postcard", "postcard"]
souvenirs3 = ["keychain", "magnet", "hat", "candy", "postcard", "stuffed bear"]

print(unique_souvenir_counts(souvenirs1))
print(unique_souvenir_counts(souvenirs2))
print(unique_souvenir_counts(souvenirs3))
```

Example Output:

```
True
Example 1 Explanation: The value "keychain" has 3 occurrences, "hat" has 2
and "postcard" has 1. No two values have the same number of occurrences.

True
Example 2 Explanation: The value "postcard" appears 4 times There's only one count (

False
Example 3 Explanation: Each item appears 1 time All counts are 1, which is not uniqu
```

▶ 💡 **Hint: Introduction to sets**

▶ 💡 **Hint: Frequency Maps**

Problem 3: Secret Beach

You make friends with a local at your latest destination, and they give you a coded message with the name of a secret beach most tourists don't know about! You are given the strings `key` and `message` which represent a cipher key and a secret message, respectively. The steps to decode the message are as follows:

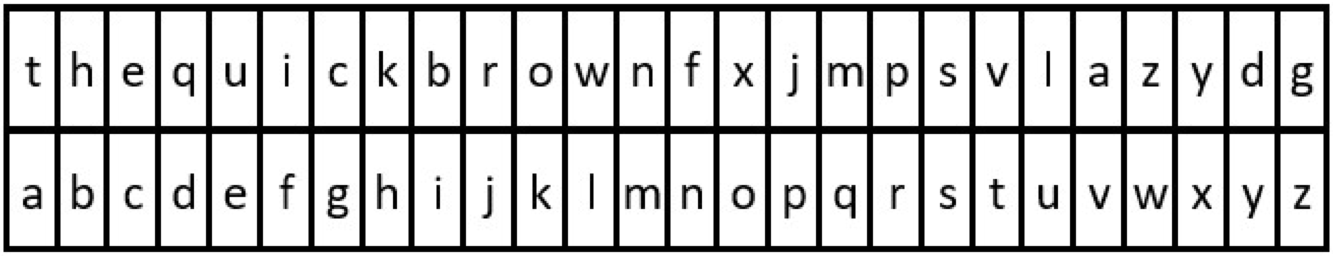
1. Use the first appearance of all 26 lowercase English letters in key as the order of the substitution table.
2. Align the substitution table with the regular English alphabet.
3. Each letter in message is then substituted using the table.
4. Spaces `' '` are transformed to themselves.

For example, given `key = "travel the world"` (an actual key would have at least one instance of each letter in the alphabet), we have the partial substitution table of `('t' -> 'a', 'r' -> 'b', 'a' -> 'c', 'v' -> 'd', 'e' -> 'e', 'l' -> 'f', 'h' -> 'g', '`
.

Write a function `decode_message()` that accepts the strings `key` and `message` and returns a string representing the decoded message.

```
def decode_message(key, message):
  pass
```

Example Usage 1:



t	h	e	q	u	i	c	k	b	r	o	w	n	f	x	j	m	p	s	v	l	a	z	y	d	g
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

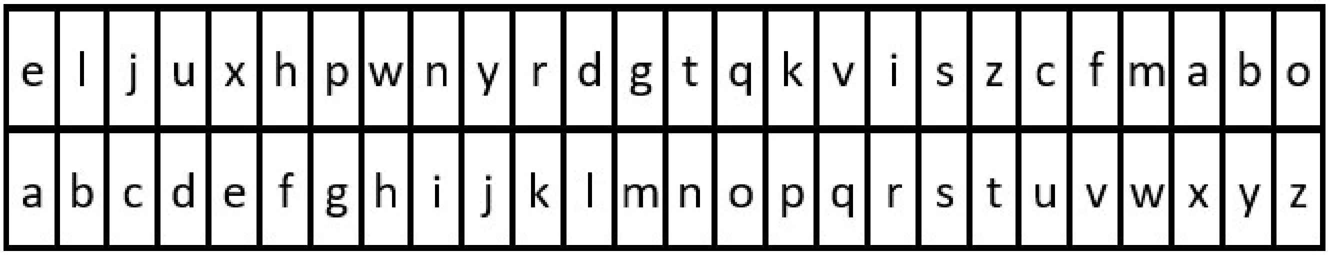
```
key1 = "the quick brown fox jumps over the lazy dog"
message1 = "vkbs bs t suepuv"

print(decode_message(key1, message1))
```

Example Output 1:

```
this is a secret
```

Example Usage 2:



e	l	j	u	x	h	p	w	n	y	r	d	g	t	q	k	v	i	s	z	c	f	m	a	b	o
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

```
key2 = "eljuxhpwnyrdgtqkviszcfmabo"
message2 = "hntu depcte lxejw lxwntu zwx piqfx"

print(decode_message(key2, message2))
```

Example Output 2:

```
find laguna beach behind the grove
```

Problem 4: Longest Harmonious Travel Sequence

In a list of travel packages, we define a harmonious travel sequence as a sequence where the difference between the maximum and minimum travel ratings is exactly 1.

Given an integer array `rating`, return the length of the longest harmonious travel sequence among all its possible subsequences.

A subsequence of an array is a sequence that can be derived from the array by deleting some or no elements without changing the order of the remaining elements.

You are provided with a partially implemented solution that contains bugs. Your task is to identify and fix the bugs to ensure the solution works correctly.

```python
def find_longest_harmonious_travel_sequence(ratings):
    # Initialize a dictionary to store the frequency of each rating
    frequency = {}

    # Count the occurrences of each rating
    for rating in ratings:
        frequency[rating] += 1

    max_length = 0

    # Find the longest harmonious sequence
    for rating in frequency:
        if rating + 1 in frequency:
            max_length = max(max_length,
                        frequency[rating] + frequency[rating - 1])

    return max_length
```

Example Usage:

```python
durations1 = [1, 3, 2, 2, 5, 2, 3, 7]
durations2 = [1, 2, 3, 4]
durations3 = [1, 1, 1, 1]

print(find_longest_harmonious_travel_sequence(durations1))
print(find_longest_harmonious_travel_sequence(durations2))
print(find_longest_harmonious_travel_sequence(durations3))
```

Example Output:

```
5
2
0
```

Problem 5: Check if All Destinations in a Route are Covered

You are given a 2D integer array `trips` and two integers `start_dest` and `end_dest`. Each `trips[i] = [starti, endi]` represents an inclusive travel interval between `starti` and `endi`.

Return `True` if each destination in the inclusive route `[start_dest, end_dest]` is covered by at least one trip in `trips`. Return `False` otherwise.

A destination `x` is covered by a trip `trips[i] = [starti, endi]` if `starti <= x <= endi`.

```
def is_route_covered(trips, start_dest, end_dest):
  pass
```

Example Usage:

```
trips1 = [[1, 2], [3, 4], [5, 6]]
start_dest1, end_dest1 = 2, 5

trips2 = [[1, 10], [10, 20]]
start_dest2, end_dest2 = 21, 21

trips3 = [[1, 2], [3, 5]]
start_dest3, end_dest1 = 2, 5

print(is_route_covered(trips1, start_dest1, end_dest1))
print(is_route_covered(trips2, start_dest2, end_dest2))
print(is_route_covered(trips3, start_dest3, end_dest3))
```

Example Output:

```
True
False
True
```

Problem 6: Most Popular Even Destination

Given a list of integers `destinations`, where each integer represents the popularity score of a destination, return the most popular even destination.

If there is a tie, return the smallest one. If there is no such destination, return `-1`.

```
def most_popular_even_destination(destinations):
  pass
```

Example Usage:

```
destinations1 = [0, 1, 2, 2, 4, 4, 1]
destinations2 = [4, 4, 4, 9, 2, 4]
destinations3 = [29, 47, 21, 41, 13, 37, 25, 7]

print(most_popular_even_destination(destinations1))
print(most_popular_even_destination(destinations2))
print(most_popular_even_destination(destinations3))
```

Example Output:

```
2
4
-1
```

Problem 7: Check if Itinerary is Valid

You are given an itinerary `itinerary` representing a list of trips between cities, where each city is represented by an integer. We consider an itinerary valid if it is a permutation of an itinerary template `base[n]`.

The template `base[n]` is defined as `[1, 2, ..., n - 1, n, n]` (in other words, it is an itinerary of length `n + 1` that visits cities `1` to `n - 1` exactly once, plus visits city `n` twice). For example, `base[1] = [1, 1]` and `base[3] = [1, 2, 3, 3]`.

Return `True` if the given itinerary is valid, otherwise return `False`.

A **permutation** is an arrangement of a set of elements. For example `[3, 2, 1]` and `[2, 3, 1]` are both possible permutations of the set of numbers `1`, `2`, and `3`.

```
def is_valid_itinerary(itinerary):
  pass
```

Example Usage:

```
itinerary1 = [2, 1, 3]
itinerary2 = [1, 3, 3, 2]
itinerary3 = [1, 1]

print(is_valid_itinerary(itinerary1))
print(is_valid_itinerary(itinerary2))
print(is_valid_itinerary(itinerary3))
```

Example Output:

```
False
Example 1 Explanation: Since the maximum element of the array is 3,
the only candidate n for which this array could be a permutation of base[n],
is n = 3. However, base[3] has four elements but array itinerary1 has three.
Therefore, it can not be a permutation of base[3] = [1, 2, 3, 3].
 So the answer is false.

True
Example 2 Explanation:  Since the maximum element of the array is 3, the only
candidate n for which this array could be a permutation of base[n], is n = 3. It
can be seen that itinerary2 is a permutation of base[3] = [1, 2, 3, 3]
(by swapping the second and fourth elements in nums, we reach base[3]).
Therefore, the answer is true.

True
Example 3 Explanation; Since the maximum element of the array is 1, the only
candidate n for which this array could be a permutation of base[n], is n = 1. It
can be seen that itinerary3 is a permutation of base[1] = [1, 1]. Therefore, the
 answer is true.
```

Problem 8: Finding Common Tourist Attractions with Least Travel Time

Given two lists of tourist attractions, `tourist_list1` and `tourist_list2`, find the common attractions with the least total travel time.

A common attraction is one that appears in both `tourist_list1` and `tourist_list2`.

A common attraction with the least total travel time is a common attraction such that if it appeared at `tourist_list1[i]` and `tourist_list2[j]` then `i + j` should be the minimum value among all the other common attractions.

Return all the common attractions with the least total travel time. Return the answer in any order.

```python
def find_attractions(tourist_list1, tourist_list2):
  pass
```

Example Usage:

```
tourist_list1 = ["Eiffel Tower","Louvre Museum","Notre-Dame","Disneyland"]
tourist_list2 = ["Colosseum","Trevi Fountain","Pantheon","Eiffel Tower"]

print(find_attractions(tourist_list1, tourist_list2))

tourist_list1 = ["Eiffel Tower","Louvre Museum","Notre-Dame","Disneyland"]
tourist_list2 = ["Disneyland","Eiffel Tower","Notre-Dame"]

print(find_attractions(tourist_list1, tourist_list2))

tourist_list1 = ["beach","mountain","forest"]
tourist_list2 = ["mountain","beach","forest"]

print(find_attractions(tourist_list1, tourist_list2))
```

Example Output:

```
["Eiffel Tower"]
["Eiffel Tower"]
["mountain", "beach"]
```

▶ 💡 **Hint: Representing Infinite Values**

Close Section

▶ Advanced Problem Set Version 1

▶ Advanced Problem Set Version 2