TIP102 | Intermediate Technical Interview Prep

Intermediate Technical Interview Prep Summer 2025 (@ Section 1b | Tuesdays and Thursdays 3PM - 5PM PDT)
Personal Member ID#: 126663

Session 1: Dictionaries

Session Overview

The focus of this session is advanced data handling in Python, focusing on functions, lists, strings, and dictionaries. Students will tackle practical programming challenges such as verifying subsequences, creating and manipulating dictionaries, and calculating values based on dynamic inputs.

The tasks are designed to enhance understanding of data structures, algorithmic thinking, and conditional logic, preparing students for more complex problem-solving scenarios involving data manipulation and retrieval.

> 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

While **each problem may call for slightly different approaches to these three steps**, the basics of the steps won't change, and it's important to engage them each time. We've built out some starting points to use in our breakout sessions, below!

Please read the following carefully (take 10 minutes as a team, if you like) and follow these basic steps, as a group, through each of the problems in your problem set.

Fun Fact: We sometimes call the main beats of problem solving, or the tasks that teams are being asked to take, our "DoNow's". If you hear a staff member using this phrase, (e.g., "Ok great! Your team is struggling with the Understand step – what might be a DoNow?") you now know what they might mean!

▼ UPI Example

Step	What is it?	Try It!
1. Understand	Here we strive to **Understand** what the interviewer is asking for. It's common to restate the problem aloud, ask questions, and consider related test cases.	• Nominate one person to share their screen so everyone is on the same page, and bring up the problem at hand. (NOTE: Please trade-off and change who is screen sharing, roughly each problem • Have one person read the problem aloud. • Have a different person restate the problem in their own words. • Have members of the group ask 2-3 questions about the problem, perhaps about the example usage, or expected output • Example: "Will the list always contain only numbers?"

2. Plan	Then we **Plan** a solution, starting with appropriate visualizations and pseudocode.	• Restate - have one person share the general idea about what the function is trying to accomplish.
		• Next, break down the problem into subproblems as a group. Each member should participate.
		  ○ If you don't know where to start, try to describe how you would solve the problem *without a computer*.
		• As a group, translate each subproblem into pseudocode.
		  ○ How do I do what I described in English in Python?
		  ○ Then, do I need to change my approach to any steps to make it work in code?

3. Implement	Now we **Implement** our solution, by translating our **Plan** into Python.	• Translate the pseudocode into Python—this is a stage at which you can consider working individually.

Breakout Problems Session 1

▾ Standard Problem Set Version 1

Problem 1: Festival Lineup

Given two lists of strings `artists` and `set_times` of length `n`, write a function `lineup()` that maps each artist to their set time.

An artist `artists[i]` has set time `set_times[i]`. Assume `i <= 0 < n` and `len(artists) == len(set_times)`.

```
def lineup(artists, set_times):
    pass
```

Example Usage:

```
artists1 = ["Kendrick Lamar", "Chappell Roan", "Mitski", "Rosalia"]
set_times1 = ["9:30 PM", "5:00 PM", "2:00 PM", "7:30 PM"]

artists2 = []
set_times2 = []

print(lineup(artists1, set_times1))
print(lineup(artists2, set_times2))
```

Example Output:

```
{"Kendrick Lamar": "9:30 PM", "Chappell Roan": "5:00 PM", "Mitski": "2:00 PM", "Rosalía": "7
{}
```

▼ ✨ **AI Hint: Dictionaries**

Key Skill: Use AI to explain code concepts

This question requires you to create a dictionary.

If you are unfamiliar with what a dictionary is, or how to create a dictionary, you can learn about Python dictionaries using a generative AI tool, like this:

"You're an expert computer science tutor. Please explain what a dictionary is in Python, and provide a simple code example of how to create one."

After you get your answer, you can also ask follow up questions:

"How is a dictionary different from a list? Can you show me examples of both?"

Problem 2: Planning App

You are designing an app for your festival to help attendees have the best experience possible! As part of the application, users will be able to easily search their favorite artist and find out the day, time, and stage the artist is playing at. Write a function `get_artist_info()` that accepts a string `artist` and a dictionary `festival_schedule` mapping artist's names to dictionaries containing the day, time, and stage they are playing on. Return the dictionary containing the information about the given `artist`.

If the artist searched for does not exist in `festival_schedule`, return the dictionary `{"message": "Artist not found"}`.

```
def get_artist_info(artist, festival_schedule):
    pass
```

Example Usage:

```
festival_schedule = {
    "Blood Orange": {"day": "Friday", "time": "9:00 PM", "stage": "Main Stage"},
    "Metallica": {"day": "Saturday", "time": "8:00 PM", "stage": "Main Stage"},
    "Kali Uchis": {"day": "Sunday", "time": "7:00 PM", "stage": "Second Stage"},
    "Lawrence": {"day": "Friday", "time": "6:00 PM", "stage": "Main Stage"}
}

print(get_artist_info("Blood Orange", festival_schedule))
print(get_artist_info("Taylor Swift", festival_schedule))
```

Example Output:

```
{'day': 'Friday', 'time': '9:00 PM', 'stage': 'Main Stage'}
{'message': 'Artist not found'}
```

▼ ✨ AI Hint: Accessing Values in a Dictionary

Key Skill: Use AI to explain code concepts

This question will require you to use keys to access their corresponding values in a dictionary. There are two common ways to access values in a dictionary. Try asking ChatGPT or GitHub copilot:

"You're an expert computer science tutor. Please show me the two most common ways to access values in a dictionary in Python, and explain how each one works."

Then open the next hint to see the answer!

▼ 💡 Hint: Dictionary Access options

The two common ways to access values in a dictionary are square bracket notation `d[key]` and the `get()` method.

The Unit 2 cheatsheet includes a more thorough breakdown of these two options. If you still feel confused after reviewing the cheatsheet, try asking generative AI to help you understand!

Problem 3: Ticket Sales

A dictionary `ticket_sales` is used to map ticket type to number of tickets sold. Return the total number of tickets of all types sold.

```
def total_sales(ticket_sales):
    pass
```

Example Usage:

```
ticket_sales = {"Friday": 200, "Saturday": 1000, "Sunday": 800, "3-Day Pass": 2500}

print(total_sales(ticket_sales))
```

Example Output:

```
4500
```

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

This question will require you to loop over a dictionary. We have three options for looping over a dictionary: looping over the keys, values, or key-value pairs. To explore how to access the keys, values, and key-value pairs reference the unit cheatsheet. For specific examples of looping over a dictionary, ask a generative AI tool to provide an example or search for existing examples using a search engine.

Problem 4: Scheduling Conflict

Demand for your festival has exceeded expectations, so you're expanding the festival to span two different venues. Some artists will perform both venues, while others will perform at just one. To ensure that there are no scheduling conflicts, implement a function `identify_conflicts()` that accepts two dictionaries `venue1_schedule` and `venue2_schedule` each mapping the artists playing at the venue to their set times. Return a dictionary containing the key-value pairs that are the same in each schedule.

```
def identify_conflicts(venue1_schedule, venue2_schedule):
    pass
```

Example Usage:

```
venue1_schedule = {
    "Stromae": "9:00 PM",
    "Janelle Monáe": "8:00 PM",
    "HARDY": "7:00 PM",
    "Bruce Springsteen": "6:00 PM"
}

venue2_schedule = {
    "Stromae": "9:00 PM",
    "Janelle Monáe": "10:30 PM",
    "HARDY": "7:00 PM",
    "Wizkid": "6:00 PM"
}

print(identify_conflicts(venue1_schedule, venue2_schedule))
```

Example Output:

```
{"Stromae": "9:00 PM", "HARDY": "7:00 PM"}
```

Problem 5: Best Set

As part of the festival, attendees cast votes for their favorite set. Given a dictionary `votes` that maps attendees id numbers to the artist they voted for, return the artist that had the most number of votes. If there is a tie, return any artist with the top number of votes.

```
def best_set(votes):
    pass
```

Example Usage:

```
votes1 = {
    1234: "SZA",
    1235: "Yo-Yo Ma",
    1236: "Ethel Cain",
    1237: "Ethel Cain",
    1238: "SZA",
    1239: "SZA"
}

votes2 = {
    1234: "SZA",
    1235: "Yo-Yo Ma",
    1236: "Ethel Cain",
    1237: "Ethel Cain",
    1238: "SZA"
}

print(best_set(votes1))
print(best_set(votes2))
```

Example Output:

```
SZA
Ethel Cain
Note: SZA and Ethel Cain would both be acceptable answers for the second example
```

▼ ✨ AI Hint: Frequency Maps

Key Skill: Use AI to explain code concepts

A dictionary that maps unique values to their frequencies within a given data structure or data type is often called a **frequency map**. Frequency maps are an extremely useful problem solving tool that you will see often throughout this unit and in future units.

We encourage you to learn by doing and attempt this problem before doing a deeper dive! However, if you get stuck, you can ask a generative AI tool like ChatGPT or GitHub Copilot to explain the concept.

For example, you could say:

"You're an expert computer science tutor for a Python-based technical interviewing course. Please explain what a frequency map is, and provide one or more examples of simple technical interview problems in which a frequency map is useful."

Problem 6: Performances with Maximum Audience

You are given an array `audiences` consisting of positive integers representing the audience size for different performances at a music festival.

Return the combined audience size of all performances in audiences with the maximum audience size.

The audience size of a performance is the number of people who attended that performance.

```
def max_audience_performances(audiences):
    pass
```

Example Usage:

```
audiences1 = [100, 200, 200, 150, 100, 250]
audiences2 = [120, 180, 220, 150, 220]

print(max_audience_performances(audiences1))
print(max_audience_performances(audiences2))
```

Example Output:

```
250
440
```

Problem 7: Performances with Maximum Audience II

If you used a dictionary as part of your solution to `max_audience_performances()` in the previous problem, try reimplementing the function without using a dictionary. If you implemented `max_audience_performances()` without using a dictionary, try solving the problem with a dictionary.

Once you've come up with your second solution, compare the two. Is one solution better than the other? Why or why not?

```
def max_audience_performances(audiences):
    pass
```

Example Usage:

```
audiences1 = [100, 200, 200, 150, 100, 250]
audiences2 = [120, 180, 220, 150, 220]

print(max_audience_performances(audiences1))
print(max_audience_performances(audiences2))
```

Example Output:

```
250
440
```

Problem 8: Popular Song Pairs

Given an array of integers `popularity_scores` representing the popularity scores of songs in a music festival playlist, return the number of popular song pairs.

A pair `(i, j)` is called popular if the songs have the same popularity score and `i < j`.

Hint: number of pairs = (n x n-1)/2

```
def num_popular_pairs(popularity_scores):
    pass
```

Example Usage:

```
popularity_scores1 = [1, 2, 3, 1, 1, 3]
popularity_scores2 = [1, 1, 1, 1]
popularity_scores3 = [1, 2, 3]

print(num_popular_pairs(popularity_scores1))
print(num_popular_pairs(popularity_scores2))
print(num_popular_pairs(popularity_scores3))
```

Example Output:

```
4
6
0
```

▼ 💡 **Hint: Floor Division**

This problem may benefit from either floor division, which is where the result of dividing two numbers is rounded down. Use a search engine or a generative AI tool to research how to perform floor division in Python.

Problem 9: Stage Arrangement Difference Between Two Performances

You are given two strings `s` and `t` representing the stage arrangements of performers in two different performances at a music festival, such that every performer occurs at most once in `s` and `t`, and `t` is a permutation of `s`.

The stage arrangement difference between `s` and `t` is defined as the sum of the absolute difference between the index of the occurrence of each performer in `s` and the index of the occurrence of the same performer in `t`.

Return the stage arrangement difference between `s` and `t`.

A **permutation** is a rearrangement of a sequence. For example, `[3, 1, 2]` and `[2, 1 , 3]` are both permutations of the list `[1, 2, 3]`.

Hint: Absolute value function

```
def find_stage_arrangement_difference(s, t):
    """
    :type s: List[str]
    :type t: List[str]
    :rtype: int
    """
```

Example Usage:

```
s1 = ["Alice", "Bob", "Charlie"]
t1 = ["Bob", "Alice", "Charlie"]
s2 = ["Alice", "Bob", "Charlie", "David", "Eve"]
t2 = ["Eve", "David", "Bob", "Alice", "Charlie"]

print(find_stage_arrangement_difference(s1, t1))
print(find_stage_arrangement_difference(s2, t2))
```

Example Output:

```
2
12
```

▼ ✨ AI Hint: Frequency Maps

Key Skill: Use AI to explain code concepts

A dictionary that maps unique values to their frequencies within a given data structure or data type is often called a **frequency map**. Frequency maps are an extremely useful problem solving tool that you will see often throughout this unit and in future units.

We encourage you to learn by doing and attempt this problem before doing a deeper dive! However, if you get stuck, you can ask a generative AI tool like ChatGPT or GitHub Copilot to explain the concept.

For example, you could say:

"You're an expert computer science tutor for a Python-based technical interviewing course. Please explain what a frequency map is, and provide one or more examples of simple technical interview problems in which a frequency map is useful."

Problem 10: VIP Passes and Guests

You're given strings `vip_passes` representing the types of guests that have VIP passes, and `guests` representing the guests you have at the music festival. Each character in guests is a type of guest you have. You want to know how many of the guests you have are also VIP pass holders.

Letters are case sensitive, so "a" is considered a different type of guest from "A".

Here is the pseudocode for the problem. Implement this in Python and explain your implementation step-by-step.

```
1. Create an empty set called vip_set.
2. For each character in vip_passes, add it to vip_set.
3. Initialize a counter variable to 0.
4. For each character in guests:
   * If the character is in vip_set, increment the count by 1.
5. Return the count.
```

```
def num_VIP_guests(vip_passes, guests):
    pass
```

Example Usage:

```
vip_passes1 = "aA"
guests1 = "aAAbbbb"

vip_passes2 = "z"
guests2 = "ZZ"

print(num_VIP_guests(vip_passes1, guests1))
print(num_VIP_guests(vip_passes2, guests2))
```

Example Output:

```
3
0
```

▼ ✨ AI Hint: Introduction to sets

Key Skill: Use AI to explain code concepts

This problem may benefit from the use of a **set**. A Python set is a data type which holds an unordered, mutable collection of *unique* elements.

If you are unfamiliar with what a set is, or how to create a set, you can learn about them using a generative AI tool, like this:

"You're an expert computer science tutor. Please explain what a set is in Python, and provide a simple code example of how to create one."

After you get your answer, you can also ask follow up questions:

"How is a set different from a list or dictionary? Can you show me examples of each?"

Problem 11: Performer Schedule Pattern

Given a string `pattern` and a string `schedule`, return `True` if `schedule` follows the same pattern. Return `False` otherwise.

Here, "follow" means a full match, such that there is a one-to-one correspondence between a letter in `pattern` and a non-empty word in `schedule`.

You are provided with a partially implemented and buggy version of the solution. Identify and fix the bugs in the code. Then, perform a thorough code review and suggest improvements.

```
def schedule_pattern(pattern, schedule):

    genres = schedule.split()

    if len(genres) == len(pattern):
        return True

    char_to_genre = {}
    genre_to_char = {}

    for char, genre in zip(pattern, genres):
        if char in char_to_genre:
            if char_to_genre[char] == genre:
                return True
        else:
            char_to_genre[char] = genre

        if genre in genre_to_char:
            if genre_to_char[genre] == char:
                return True
        else:
            genre_to_char[genre] = char

    return False
```

Example Usage:

```
pattern1 = "abba"
schedule1 = "rock jazz jazz rock"

pattern2 = "abba"
schedule2 = "rock jazz jazz blues"

pattern3 = "aaaa"
schedule3 = "rock jazz jazz rock"

print(schedule_pattern(pattern1, schedule1))
print(schedule_pattern(pattern2, schedule2))
print(schedule_pattern(pattern3, schedule3))
```

Example Output:

```
True
False
False
```

▼ ✨ AI Hint: `zip()`

Key Skill: Use AI to explain code concepts

This problem may benefit from use of the `zip()` function. For a quick refresher on how the `zip()` function works, check out the Unit 2 Cheatsheet.

If you'd still like to see more examples or ask follow-up questions, try using an AI tool like ChatGPT or GitHub Copilot. You can use the following prompt as a starting point:

"You're an expert computer science tutor. Please provide 2-3 examples of how the `zip()` function is used in Python, and explain how each one works."

Problem 12: Sort the Performers

You are given an array of strings `performer_names`, and an array `performance_times` that consists of distinct positive integers representing the performance durations in minutes. Both arrays are of length `n`.

For each index `i`, `performer_names[i]` and `performance_times[i]` denote the name and performance duration of the `i`th performer.

Return `performer_names` sorted in descending order by the performance durations.

```
def sort_performers(performer_names, performance_times):
    """
    :type performer_names: List[str]
    :type performance_times: List[int]
    :rtype: List[str]
    """
```

Example Usage:

```
performer_names1 = ["Mary", "John", "Emma"]
performance_times1 = [180, 165, 170]

performer_names2 = ["Alice", "Bob", "Bob"]
performance_times2 = [155, 185, 150]

print(sort_performers(performer_names1, performance_times1))
print(sort_performers(performer_names2, performance_times2))
```

Example Output:

```
["Mary", "Emma", "John"]
["Bob", "Alice", "Bob"]
```

▼ ✨ **AI Hint: `sorted()`**

Key Skill: Use AI to explain code concepts

This problem may benefit from use of the `sorted()` function. For a quick refresher on how the `sorted()` function works, check out the Unit 2 Cheatsheet.

If you'd still like to see more examples or ask follow-up questions, try using an AI tool like ChatGPT or GitHub Copilot. You can use the following prompt as a starting point:

"You're an expert computer science tutor. Please provide 2-3 examples of how the `sorted()` function is used in Python, and explain how each one works."

Close Section

▾ Standard Problem Set Version 2

Problem 1: Space Crew

Given two lists of length `n`, `crew` and `position`, map the space station crew to their position on board the international space station.

Each crew member `crew[i]` has job `position[i]` on board, where `0 <= i < n` and `len(crew) == len(position)`.

Hint: Introduction to dictionaries

```
def space_crew(crew, position):
    pass
```

Example Usage:

```
exp70_crew = ["Andreas Mogensen", "Jasmin Moghbeli", "Satoshi Furukawa", "Loral O'Hara", "Ko
exp70_positions = ["Commander", "Flight Engineer", "Flight Engineer", " Flight Engineer", "F

ax3_crew = ["Michael Lopez-Alegria", "Walter Villadei", "Alper Gezeravci", "Marcus Wandt"]
ax3_positions = ["Commander", "Mission Pilot", "Mission Specialist", "Mission Specialist"]

print(space_crew(exp70_crew, exp70_positions))
print(space_crew(ax3_crew, ax3_positions))
```

Example Output:

```
{
    "Andreas Mogensen": "Commander",
    "Jasmin Moghbeli": "Flight Engineer",
    "Satoshi Furukawa": "Flight Engineer",
    "Loral O'Hara": "Flight Engineer",
    "Konstantin Borisov": "Flight Engineer",
}

{
    "Michael López-Alegría": "Commander",
    "Walter Villadei": "Mission Pilot",
    "Alper Gezeravcı": "Mission Specialist",
    "Marcus Wandt": "Mission Specialist"
}
```

▼ ✨ AI Hint: Dictionaries

Key Skill: Use AI to explain code concepts

This question requires you to create a dictionary.

If you are unfamiliar with what a dictionary is, or how to create a dictionary, you can learn about Python dictionaries using a generative AI tool, like this:

"You're an expert computer science tutor. Please explain what a dictionary is in Python, and provide a simple code example of how to create one."

After you get your answer, you can also ask follow up questions:

"How is a dictionary different from a list? Can you show me examples of both?"

Problem 2: Space Encyclopedia

Given a dictionary `planets` that maps planet names to a dictionary containing the planet's number of moons and orbital period, write a function `planet_lookup()` that accepts a string `planet_name` and returns a string in the form `Planet <planet_name> has an orbital period of <orbital period> Earth days and has <number of mo`
If `planet_name` is not a key in `planets`, return `"Sorry, I have no data on that planet."`.

```
def planet_lookup(planet_name):
    pass
```

Example Usage:

```
planetary_info = {
    "Mercury": {
        "Moons": 0,
        "Orbital Period": 88
    },
    "Earth": {
        "Moons": 1,
        "Orbital Period": 365.25
    },
    "Mars": {
        "Moons": 2,
        "Orbital Period": 687
    },
    "Jupiter": {
        "Moons": 79,
        "Orbital Period": 10592
    }
}

print(planet_lookup("Jupiter"))
print(planet_lookup("Pluto"))
```

Example Output:

```
Planet Jupiter has an orbital period of 10592 Earth days and has 79 moons.
Sorry, I have no data on that planet.
```

▼ ✨ AI Hint: Accessing Values in a Dictionary

Key Skill: Use AI to explain code concepts

This question will require you to use keys to access their corresponding values in a dictionary. There are two common ways to access values in a dictionary. Try asking ChatGPT or GitHub copilot:

"You're an expert computer science tutor. Please show me the two most common ways to access values in a dictionary in Python, and explain how each one works."

Then open the next hint to see the answer!

▼ 💡 Hint: Dictionary Access options

The two common ways to access values in a dictionary are square bracket notation `d[key]` and the `get()` method.

The Unit 2 cheatsheet includes a more thorough breakdown of these two options. If you still feel confused after reviewing the cheatsheet, try asking generative AI to help you understand!

▼ 💡 **Nested Data**

This problem makes use of nested dictionaries. To learn more about nested dictionaries and other nested data structures, check out the unit cheatsheet.

Problem 3: Breathing Room

As part of your job as an astronaut, you need to perform routine safety checks. You are given a dictionary `oxygen_levels` which maps room names to current oxygen levels and two integers `min_val` and `max_val` specifying the acceptable range of oxygen levels. Return a list of room names whose values are outside the range defined by `min_val` and `max_val` (inclusive).

```
def check_oxygen_levels(oxygen_levels, min_val, max_val):
    pass
```

Example Usage:

```
oxygen_levels = {
    "Command Module": 21,
    "Habitation Module": 20,
    "Laboratory Module": 19,
    "Airlock": 22,
    "Storage Bay": 18
}

min_val = 19
max_val = 22

print(check_oxygen_levels(oxygen_levels, min_val, max_val))
```

Example Output:

```
['Storage Bay']
```

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

> This question will require you to loop over a dictionary. We have three options for looping over a dictionary: looping over the keys, values, or key-value pairs. To explore how to access the keys, values, and key-value pairs reference the unit cheatsheet. For specific examples of looping over a dictionary, ask a generative AI tool to provide an example or search for existing examples using a search engine.

Problem 4: Experiment Analysis

Write a function `data_difference()` that accepts two dictionaries `experiment1` and `experiment2` and returns a new dictionary that contains only key-value pairs found exclusively in `experiment1` but not in `experiment2`.

```
def data_difference(experiment1, experiment2):
    pass
```

Example Usage:

```
exp1_data = {'temperature': 22, 'pressure': 101.3, 'humidity': 45}
exp2_data = {'temperature': 18, 'pressure': 101.3, 'radiation': 0.5}

print(data_difference(exp1_data, exp2_data))
```

Example Output:

```
{'temperature': 22, 'humidity': 45}
```

Problem 5: Name the Node

NASA has asked the public to vote on a new name for one of the nodes in the International Space Station. Given a list of strings `votes` where each string in the list is a voter's suggested new name, implement a function `get_winner()` that returns the suggestion with the most number of votes.

If there is a tie, return either option.

```
def get_winner(votes):
    pass
```

Example Usage:

```
votes1 = ["Colbert", "Serenity", "Serenity", "Tranquility", "Colbert", "Colbert"]
votes2 = ["Colbert", "Serenity", "Serenity", "Tranquility", "Colbert"]

print(get_winner(votes1))
print(get_winner(votes2))
```

Example Output:

```
Colbert
Serenity

Note: Colbert and Serenity would both be acceptable answers for the second example
```

▼ ✨ AI Hint: Frequency Maps

Key Skill: Use AI to explain code concepts

A dictionary that maps unique values to their frequencies within a given data structure or data type is often called a **frequency map**. Frequency maps are an extremely useful problem solving tool that you will see often throughout this unit and in future units.

We encourage you to learn by doing and attempt this problem before doing a deeper dive! However, if you get stuck, you can ask a generative AI tool like ChatGPT or GitHub Copilot to explain the concept.

For example, you could say:

"You're an expert computer science tutor for a Python-based technical interviewing course. Please explain what a frequency map is, and provide one or more examples of simple technical interview problems in which a frequency map is useful."

Problem 6: Check if the Transmission is Complete

Ground control has sent a transmission containing important information. A complete transmission is one where every letter of the English alphabet appears at least once.

Given a string `transmission` containing only lowercase English letters, return `true` if the transmission is complete, or `false` otherwise.

```
def check_if_complete_transmission(transmission):
    """
    :type transmission: str
    :rtype: bool
    """
```

Example Usage:

```
transmission1 = "thequickbrownfoxjumpsoverthelazydog"
transmission2 = "spacetravel"

print(check_if_complete_transmission(transmission1))
print(check_if_complete_transmission(transmission2))
```

Example Output:

```
True
False
```

Problem 7: Signal Pairs

Ground control is analyzing signal patterns received from different probes. You are given a 0-indexed array `signals` consisting of distinct strings.

The string `signals[i]` can be paired with the string `signals[j]` if the string `signals[i]` is equal to the reversed string of `signals[j]`. `0 <= i < j < len(signals)`. Return the maximum number of pairs that can be formed from the array `signals`.

Note that each string can belong in at most one pair.

```
def max_number_of_string_pairs(signals):
    pass
```

Example Usage:

```
signals1 = ["cd", "ac", "dc", "ca", "zz"]
signals2 = ["ab", "ba", "cc"]
signals3 = ["aa", "ab"]

print(max_number_of_string_pairs(signals1))
print(max_number_of_string_pairs(signals2))
print(max_number_of_string_pairs(signals3))
```

Example Output:

```
2
1
0
```

Problem 8: Find the Difference of Two Signal Arrays

You are given two 0-indexed integer arrays `signals1` and `signals2`, representing signal data from two different probes. Return a list `answer` of size 2 where:

- `answer[0]` is a list of all distinct integers in `signals1` which are not present in `signals2`.
- `answer[1]` is a list of all distinct integers in `signals2` which are not present in `signals1`.

Note that the integers in the lists may be returned in any order.

Below is the pseudocode for the problem. Implement this in Python and explain your implementation step-by-step.

```
1. Convert signals1 and signals2 to sets.
2. Find the difference between set1 and set2 and store it in diff1.
3. Find the difference between set2 and set1 and store it in diff2.
4. Return the list [diff1, diff2].
```

```
def find_difference(signals1, signals2):
    pass
```

Example Usage:

```
signals1_example1 = [1, 2, 3]
signals2_example1 = [2, 4, 6]

signals1_example2 = [1, 2, 3, 3]
signals2_example2 = [1, 1, 2, 2]

print(find_difference(signals1_example1, signals2_example1))
print(find_difference(signals1_example2, signals2_example2))
```

Example Output:

```
[[1, 3], [4, 6]]
[[3], []]
```

▼ ✨ AI Hint: Introduction to sets

Key Skill: Use AI to explain code concepts

This problem may benefit from the use of a **set**. A Python set is a data type which holds an unordered, mutable collection of *unique* elements.

If you are unfamiliar with what a set is, or how to create a set, you can learn about them using a generative AI tool, like this:

"You're an expert computer science tutor. Please explain what a set is in Python, and provide a simple code example of how to create one."

After you get your answer, you can also ask follow up questions:

"How is a set different from a list or dictionary? Can you show me examples of each?"

Problem 9: Common Signals Between Space Probes

Two space probes have collected signals represented by integer arrays `signals1` and `signals2` of sizes `n` and `m`, respectively. Calculate the following values:

- `answer1`: the number of indices `i` such that `signals1[i]` exists in `signals2`.

- `answer2`: the number of indices `j` such that `signals2[j]` exists in `signals1`.

Return `[answer1, answer2]`.

```
def find_common_signals(signals1, signals2):
    pass
```

Example Usage:

```
signals1_example1 = [2, 3, 2]
signals2_example1 = [1, 2]
print(find_common_signals(signals1_example1, signals2_example1))

signals1_example2 = [4, 3, 2, 3, 1]
signals2_example2 = [2, 2, 5, 2, 3, 6]
print(find_common_signals(signals1_example2, signals2_example2))

signals1_example3 = [3, 4, 2, 3]
signals2_example3 = [1, 5]
print(find_common_signals(signals1_example3, signals2_example3))
```

Example Output:

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

Problem 10: Common Signals Between Space Probes II

If you implemented `find_common_signals()` in the previous problem using dictionaries, try implementing `find_common_signals()` again using sets instead of dictionaries. If you implemented `find_common_signals()` using sets, use dictionaries this time.

Once you've come up with your second solution, compare the two. Is one solution better than the other? How so? Why or why not?

```
def find_common_signals(signals1, signals2):
    pass
```

Example Usage:

```
signals1_example1 = [2, 3, 2]
signals2_example1 = [1, 2]
print(find_common_signals(signals1_example1, signals2_example1))

signals1_example2 = [4, 3, 2, 3, 1]
signals2_example2 = [2, 2, 5, 2, 3, 6]
print(find_common_signals(signals1_example2, signals2_example2))

signals1_example3 = [3, 4, 2, 3]
signals2_example3 = [1, 5]
print(find_common_signals(signals1_example3, signals2_example3))
```

Example Output:

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

Problem 11: Sort Signal Data

Ground control needs to analyze the frequency of signal data received from different probes. Given an array of integers `signals`, sort the array in increasing order based on the frequency of the values. If multiple values have the same frequency, sort them in decreasing order. Return the sorted array.

Below is a buggy or incomplete version of the solution. Identify and fix the bugs in the code. Then, perform a code review and suggest improvements.

```
def frequency_sort(signals):
    freq = {}
    for signal in signals:
        if signal in freq:
            freq[signal] += 1
        else:
            freq[signal] = 0

    sorted_signals = sorted(signals, key=lambda x: (freq[x], x))

    return sorted_signals
```

Example Usage:

```
signals1 = [1, 1, 2, 2, 2, 3]
signals2 = [2, 3, 1, 3, 2]
signals3 = [-1, 1, -6, 4, 5, -6, 1, 4, 1]

print(frequency_sort(signals1))
print(frequency_sort(signals2))
print(frequency_sort(signals3))
```

Example Output:

```
[3, 1, 1, 2, 2, 2]
[1, 3, 3, 2, 2]
[5, -1, 4, 4, -6, -6, 1, 1, 1]
```

▼ ✨ **AI Hint: `sorted()`**

Key Skill: Use AI to explain code concepts

This problem may benefit from use of the `sorted()` function. For a quick refresher on how the `sorted()` function works, check out the Unit 2 Cheatsheet.

If you'd still like to see more examples or ask follow-up questions, try using an AI tool like ChatGPT or GitHub Copilot. You can use the following prompt as a starting point:

"You're an expert computer science tutor. Please provide 2-3 examples of how the `sorted()` function is used in Python, and explain how each one works."

▼ 💡 **Hint: Lambda Functions**

This problem may benefit from understanding lambda functions. To learn more about lambda functions, conduct your own independent research or check out the advanced section of the unit cheatsheet.

Problem 12: Final Communication Hub

You are given an array `paths`, where `paths[i] = [hubA, hubB]` means there exists a direct communication path going from `hubA` to `hubB`. Return the final communication hub, that is, the hub without any outgoing path to another hub.

It is guaranteed that the paths form a line without any loops, therefore, there will be exactly one final communication hub.

```
def find_final_hub(paths):
    pass
```

Example Usage:

```
paths1 = [["Earth", "Mars"], ["Mars", "Titan"], ["Titan", "Europa"]]
paths2 = [["Alpha", "Beta"], ["Gamma", "Alpha"], ["Beta", "Delta"]]
paths3 = [["StationA", "StationZ"]]

print(find_final_hub(paths1))
print(find_final_hub(paths2))
print(find_final_hub(paths3))
```

Example Output:

```
"Europa"
"Delta"
"StationZ"
```

Close Section

▾ Advanced Problem Set Version 1

Problem 1: Counting Treasure

Captain Blackbeard has a treasure map with several clues that point to different locations on an island. Each clue is associated with a specific location and the number of treasures buried there. Given a dictionary `treasure_map` where keys are location names and values are integers representing the number of treasures buried at those locations, write a function `total_treasures()` that returns the total number of treasures buried on the island.

```
def total_treasure(treasure_map):
    pass
```

Example Usage:

```
treasure_map1 = {
    "Cove": 3,
    "Beach": 7,
    "Forest": 5
}

treasure_map2 = {
    "Shipwreck": 10,
    "Cave": 20,
    "Lagoon": 15,
    "Island Peak": 5
}

print(total_treasures(treasure_map1))
print(total_treasures(treasure_map2))
```

Example Output:

```
15
50
```

▼ ✨ AI Hint: Dictionaries

Key Skill: Use AI to explain code concepts

This question requires you to create a dictionary.

If you are unfamiliar with what a dictionary is, or how to create a dictionary, you can learn about Python dictionaries using a generative AI tool, like this:

"You're an expert computer science tutor. Please explain what a dictionary is in Python, and provide a simple code example of how to create one."

After you get your answer, you can also ask follow up questions:

"How is a dictionary different from a list? Can you show me examples of both?"

▼ ✨ AI Hint: Accessing Values in a Dictionary

Key Skill: Use AI to explain code concepts

This question will require you to use keys to access their corresponding values in a dictionary. There are two common ways to access values in a dictionary. Try asking ChatGPT or GitHub copilot:

"You're an expert computer science tutor. Please show me the two most common ways to access values in a dictionary in Python, and explain how each one works."

Then open the next hint to see the answer!

▼ 💡 Hint: Dictionary Access options

The two common ways to access values in a dictionary are square bracket notation `d[key]` and the `get()` method.

The Unit 2 cheatsheet includes a more thorough breakdown of these two options. If you still feel confused after reviewing the cheatsheet, try asking generative AI to help you understand!

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

This question will require you to loop over a dictionary. We have three options for looping over a dictionary: looping over the keys, values, or key-value pairs. To explore how to access the keys, values, and key-value pairs reference the unit cheatsheet. For specific examples of looping over a dictionary, ask a generative AI tool to provide an example or search for existing examples using a search engine.

Problem 2: Pirate Message Check

Taken captive, Captain Anne Bonny has been smuggled a secret message from her crew. She will know she can trust the message if it contains all of the letters in the alphabet. Given a string `message` containing only lowercase English letters and whitespace, write a function `can_trust_message()` that returns `True` if the message contains every letter of the English alphabet at least once, and `False` otherwise.

```
def can_trust_message(message):
    pass
```

Example Usage:

```
message1 = "sphinx of black quartz judge my vow"
message2 = "trust me"

print(can_trust_message(message1))
print(can_trust_message(message2))
```

Example Output:

```
True
False
```

▼ ✨ **AI Hint: Introduction to sets**

Key Skill: Use AI to explain code concepts

This problem may benefit from the use of a **set**. A Python set is a data type which holds an unordered, mutable collection of *unique* elements.

If you are unfamiliar with what a set is, or how to create a set, you can learn about them using a generative AI tool, like this:

"You're an expert computer science tutor. Please explain what a set is in Python, and provide a simple code example of how to create one."

After you get your answer, you can also ask follow up questions:

"How is a set different from a list or dictionary? Can you show me examples of each?"

Problem 3: Find All Duplicate Treasure Chests in an Array

Captain Blackbeard has an integer array `chests` of length `n` where all the integers in `chests` are in the range `[1, n]` and each integer appears once or twice. Return an array of all the integers that appear twice, representing the treasure chests that have duplicates.

```
def find_duplicate_chests(chests):
    pass
```

Example Usage:

```
chests1 = [4, 3, 2, 7, 8, 2, 3, 1]
chests2 = [1, 1, 2]
chests3 = [1]

print(find_duplicate_chests(chests1))
print(find_duplicate_chests(chests2))
print(find_duplicate_chests(chests3))
```

Example Output:

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

▼ ✨ AI Hint: Frequency Maps

Key Skill: Use AI to explain code concepts

A dictionary that maps unique values to their frequencies within a given data structure or data type is often called a **frequency map**. Frequency maps are an extremely useful problem solving tool that you will see often throughout this unit and in future units.

We encourage you to learn by doing and attempt this problem before doing a deeper dive! However, if you get stuck, you can ask a generative AI tool like ChatGPT or GitHub Copilot to explain the concept.

For example, you could say:

"You're an expert computer science tutor for a Python-based technical interviewing course. Please explain what a frequency map is, and provide one or more examples of simple technical interview problems in which a frequency map is useful."

Problem 4: Booby Trap

Captain Feathersword has found another pirate's buried treasure, but they suspect it's booby-trapped. The treasure chest has a secret code written in pirate language, and Captain Feathersword believes the trap can be disarmed if the code can be balanced. A balanced code is one where the frequency of every letter present in the code is equal. To disable the trap, Captain Feathersword *must* remove exactly one letter from the message. Help Captain Feathersword determine if it's possible to remove one letter to balance the pirate code.

Given a 0-indexed string `code` consisting of only lowercase English letters, write a function `is_balanced()` that returns `True` if it's possible to remove one letter so that the frequency of all remaining letters is equal, and `False` otherwise.

```
def is_balanced(code):
    pass
```

Example Usage:

```
code1 = "arghh"
code2 = "haha"

print(is_balanced(code1))
print(is_balanced(code2))
```

Example Output:

```
True
Explanation: Select index 4 and delete it: word becomes "argh" and each character has a frequ

False
Explanation: They must delete a character, so either the frequency of "h" is 1 and the freque
```

Problem 5: Overflowing With Gold

Captain Feathersword and their crew has discovered a list of gold amounts at various hidden locations on an island. Each number on the map corresponds to the amount of gold at a specific location. Captain Feathersword already has plenty of loot, and their ship is nearly full. They want to find two distinct locations on the map such that the sum of the gold amounts at these two locations is exactly equal to the amount of space left on their ship.

Given an array of integers `gold_amounts` representing the amount of gold at each location and an integer `target`, return the *indices* of the two locations whose gold amounts add up to the target.

Assume that each input has exactly one solution, and you may not use the same location twice. You can return the answer in any order.

```
def find_treasure_indices(gold_amounts, target):
    pass
```

Example Usage:

```
gold_amounts1 = [2, 7, 11, 15]
target1 = 9

gold_amounts2 = [3, 2, 4]
target2 = 6

gold_amounts3 = [3, 3]
target3 = 6

print(find_treasure_indices(gold_amounts1, target1))
print(find_treasure_indices(gold_amounts2, target2))
print(find_treasure_indices(gold_amounts3, target3))
```

Example Output:

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

Problem 6: Organize the Pirate Crew

Captain Blackbeard needs to organize his pirate crew into different groups for a treasure hunt. Each pirate has a unique ID from 0 to n - 1.

You are given an integer array `group_sizes`, where `group_sizes[i]` is the size of the group that pirate `i` should be in. For example, if `group_sizes[1] = 3`, then pirate 1 must be in a group of size 3.

Return a list of groups such that each pirate `i` is in a group of size `group_sizes[i]`.

Each pirate should appear in exactly one group, and every pirate must be in a group. If there are multiple answers, return any of them. It is guaranteed that there will be at least one valid solution for the given input.

```
def organize_pirate_crew(group_sizes):
    pass
```

Example Usage:

```
group_sizes1 = [3, 3, 3, 3, 3, 1, 3]
group_sizes2 = [2, 1, 3, 3, 3, 2]

print(organize_pirate_crew(group_sizes1))
print(organize_pirate_crew(group_sizes2))
```

Example Output:

```
[[5], [0, 1, 2], [3, 4, 6]]
[[1], [0, 5], [2, 3, 4]]
```

Problem 7: Minimum Number of Steps to Match Treasure Maps

Captain Blackbeard has two treasure maps represented by two strings of the same length `map1` and `map2`. In one step, you can choose any character of `map2` and replace it with another character.

Return the minimum number of steps to make `map2` an anagram of `map1`.

An Anagram of a string is a string that contains the same characters with a different (or the same) ordering.

```
def min_steps_to_match_maps(map1, map2):
    pass
```

Example Usage:

```
map1_1 = "bab"
map2_1 = "aba"
map1_2 = "treasure"
map2_2 = "huntgold"
map1_3 = "anagram"
map2_3 = "mangaar"

print(min_steps_to_match_maps(map1_1, map2_1))
print(min_steps_to_match_maps(map1_2, map2_2))
print(min_steps_to_match_maps(map1_3, map2_3))
```

Example Output:

```
1
6
0
```

Problem 8: Counting Pirates' Action Minutes

Captain Dread is keeping track of the crew's activities using a log. The logs are represented by a 2D integer array `logs` where each `logs[i] = [pirateID, time]` indicates that the pirate with `pirateID` performed an action at the minute `time`.

Multiple pirates can perform actions simultaneously, and a single pirate can perform multiple actions in the same minute.

The pirate action minutes (PAM) for a given pirate is defined as the number of unique minutes in which the pirate performed an action. A minute can only be counted once, even if multiple actions occur during it.

You are to calculate a 1-indexed array `answer` of size `k` such that, for each `j (1 <= j <= k)`, `answer[j]` is the number of pirates whose PAM equals `j`.

Return the array `answer` as described above.

```
def counting_pirates_action_minutes(logs, k):
    pass
```

Example Usage:

```
logs1 = [[0, 5], [1, 2], [0, 2], [0, 5], [1, 3]]
k1 = 5
logs2 = [[1, 1], [2, 2], [2, 3]]
k2 = 4

print(counting_pirates_action_minutes(logs1, k1))
print(counting_pirates_action_minutes(logs2, k2))
```

Example Output:

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

Close Section

▾ Advanced Problem Set Version 2

Problem 1: The Library of Alexandria

In the ancient Library of Alexandria, a temporal rift has scattered several important scrolls across different rooms. You are given a dictionary `library_catalog` that maps room names to the number of scrolls that room should have and a second dictionary `actual_distribution` that maps room names to the number of scrolls found in that room after the temporal rift.

Write a function `analyze_library()` that determines if any room has more or fewer scrolls than it should. The function should return a dictionary where the keys are the room names and the values are the differences in the number of scrolls (actual number of scrolls - expected number of scrolls).

You must loop over the dictionaries to compute the differences.

```
def analyze_library(library_catalog, actual_distribution):
    pass
```

Example Usage:

```
library_catalog = {
    "Room A": 150,
    "Room B": 200,
    "Room C": 250,
    "Room D": 300
}

actual_distribution = {
    "Room A": 150,
    "Room B": 190,
    "Room C": 260,
    "Room D": 300
}


print(analyze_library(library_catalog, actual_distribution))
```

Example Output:

```
{'Room A': 0, 'Room B': -10, 'Room C': 10, 'Room D': 0}
```

▼ ✨ AI Hint: Dictionaries

Key Skill: Use AI to explain code concepts

This question requires you to create a dictionary.

If you are unfamiliar with what a dictionary is, or how to create a dictionary, you can learn about Python dictionaries using a generative AI tool, like this:

"You're an expert computer science tutor. Please explain what a dictionary is in Python, and provide a simple code example of how to create one."

After you get your answer, you can also ask follow up questions:

"How is a dictionary different from a list? Can you show me examples of both?"

▼ ✨ AI Hint: Accessing Values in a Dictionary

Key Skill: Use AI to explain code concepts

This question will require you to use keys to access their corresponding values in a dictionary. There are two common ways to access values in a dictionary. Try asking ChatGPT or GitHub copilot:

"You're an expert computer science tutor. Please show me the two most common ways to access values in a dictionary in Python, and explain how each one works."

Then open the next hint to see the answer!

▼ 💡 **Hint: Dictionary Access options**

The two common ways to access values in a dictionary are square bracket notation `d[key]` and the `get()` method.

The Unit 2 cheatsheet includes a more thorough breakdown of these two options. If you still feel confused after reviewing the cheatsheet, try asking generative AI to help you understand!

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

This question will require you to loop over a dictionary. We have three options for looping over a dictionary: looping over the keys, values, or key-value pairs. To explore how to access the keys, values, and key-value pairs reference the unit cheatsheet. For specific examples of looping over a dictionary, ask a generative AI tool to provide an example or search for existing examples using a search engine.

Problem 2: Grecian Artifacts

You've spent your last few trips exploring different periods of Ancient Greece. During your travels, you discover several interesting artifacts. Some artifacts appear in multiple time periods, while others in just one.

You are given two lists of strings `artifacts1` and `artifacts2` representing the artifacts found in two different time periods. Write a function `find_common_artifacts()` that returns a list of artifacts common to both time periods.

```
def find_common_artifacts(artifacts1, artifacts2):
    pass
```

Example Usage:

```
artifacts1 = ["Statue of Zeus", "Golden Vase", "Bronze Shield"]
artifacts2 = ["Golden Vase", "Silver Sword", "Bronze Shield"]

print(find_common_artifacts(artifacts1, artifacts2))
```

Example Output:

```
 ["Golden Vase", "Bronze Shield"]
```

▼ ✨ AI Hint: Introduction to sets

Key Skill: Use AI to explain code concepts

This problem may benefit from the use of a **set**. A Python set is a data type which holds an unordered, mutable collection of *unique* elements.

If you are unfamiliar with what a set is, or how to create a set, you can learn about them using a generative AI tool, like this:

"You're an expert computer science tutor. Please explain what a set is in Python, and provide a simple code example of how to create one."

After you get your answer, you can also ask follow up questions:

"How is a set different from a list or dictionary? Can you show me examples of each?"

Problem 3: Souvenir Declutter

As a time traveler, you've collected a mountain of souvenirs over the course of your travels. You're running out of room to store them all and need to declutter. Given a list of strings `souvenirs` and a integer `threshold`, declutter your souvenirs by writing a function `declutter()` that returns a list of souvenirs strictly below `threshold`.

```
def declutter(souvenirs, threshold):
    pass
```

Example Usage:

```
souvenirs1 = ["coin", "alien egg", "coin", "coin", "map", "map", "statue"]
threshold1 = 3

souvenirs2 = ["postcard", "postcard", "postcard", "sword"]
threshold = 2
```

Example Output:

```
["alien egg", "map", "map", "statue"]
["sword"]
```

▼ ✨ AI Hint: Frequency Maps

Key Skill: Use AI to explain code concepts

A dictionary that maps unique values to their frequencies within a given data structure or data type is often called a **frequency map**. Frequency maps are an extremely useful problem solving tool that you will see often throughout this unit and in future units.

We encourage you to learn by doing and attempt this problem before doing a deeper dive! However, if you get stuck, you can ask a generative AI tool like ChatGPT or GitHub Copilot to explain the concept.

For example, you could say:

"You're an expert computer science tutor for a Python-based technical interviewing course. Please explain what a frequency map is, and provide one or more examples of simple technical interview problems in which a frequency map is useful."

Problem 4: Time Portals

In your time travel adventures, you are given an array of digit strings `portals` and a digit string `destination`. Return the number of pairs of indices `(i, j)` (where `i != j`) such that the concatenation of `portals[i] + portals[j]` equals `destination`.

***Note**: For index values `i` and `j`, the pairs `(i, j)` and `(j, i)` are considered different - order matters.*

```
def num_of_time_portals(portals, destination):
    pass
```

Example Usage:

```
portals1 = ["777", "7", "77", "77"]
destination1 = "7777"
portals2 = ["123", "4", "12", "34"]
destination2 = "1234"
portals3 = ["1", "1", "1"]
destination3 = "11"

print(num_of_time_portals(portals1, destination1))
print(num_of_time_portals(portals2, destination2))
print(num_of_time_portals(portals3, destination3))
```

Example Output:

```
4
2
6
```

Problem 5: Detect Temporal Anomaly

As a time traveler, you have recorded the occurrences of specific events at different time points. You suspect that some events might be occurring too frequently within short time spans, indicating potential temporal anomalies. Given an array `time_points` where each element represents an event ID at a particular time point, and an integer `k`, determine if there are two distinct time points `i` and `j` such that `time_points[i] == time_points[j]` and the absolute difference between `i` and `j` is at most `k`.

***Note**: The indices must be unique, but not the values `i` and `j` themselves.*

```
def detect_temporal_anomaly(time_points, k):
    pass
```

Example Usage:

```


time_points1 = [1, 2, 3, 1]
k1 = 3

time_points2 = [1, 0, 1, 1]
k2 = 1

time_points3 = [1, 2, 3, 1, 2, 3]
k3 = 2

print(detect_temporal_anomaly(time_points1, k1))
print(detect_temporal_anomaly(time_points2, k2))
print(detect_temporal_anomaly(time_points3, k3))
```

Example Output:

```
True
True
False
```

Problem 6: Time Portal Race Rankings

A group of time travelers are competing in a series of races to see who can hop through time portals the fastest, from the medieval era to the year 4050.

You're given a list of race outcomes in the form of an integer array `races`, where each element `races[i] = [winner, loser]` indicates that the traveler `winner` defeated the traveler `loser` in a race.

Write a function `find_travelers()` that accepts the integer array `races` as a parameter and returns a list `answer` of length 2 where:

`answer[0]` is a list of all travelers who have not lost any races.

`answer[1]` is a list of all travelers who have lost exactly one race.

Both the input list `races` and your output list `answer` should be sorted in **increasing order.**

Note: Only include travelers who have competed in at least one race — that is, those who appear as either a winner or a loser in the input list `races`. For example, if `races = [[1,2], [5, 6]`, that may imply the existence of racers `3` and `4`. However, since neither racer `3` nor `4` is included in the input list, `3` and `4` should also not appear in the output list `answer`.

```
def find_travelers(races):
    pass
```

Example Usage:

```
races1 = [[1, 3], [2, 3], [3, 6], [5, 6], [5, 7], [4, 5], [4, 8], [4, 9], [10, 4], [10, 9]]
races2 = [[2, 3], [1, 3], [5, 4], [6, 4]]

print(find_travelers(races1))
print(find_travelers(races2))
```

Example Output

```
[[1, 2, 10], [4, 5, 7, 8]]
[[1, 2, 5, 6], []]
```

Problem 7: Lingual Frequencies

As a time traveling linguist, you are analyzing texts written in an ancient script. However, some words in the text are illegible and can't be deciphered. Write a function `find_most_frequent_word()` that accepts a string `text` and a list of illegible words `illegibles` and returns the most frequent word in `text` that is not an illegible word.

```
def find_most_frequent_word(text, illegibles):
    pass
```

Example Usage:

```
paragraph1 = "a."
illegibles1 = []
print(find_most_frequent_word(paragraph1, illegibles1))

paragraph2 = "Bob hit a ball, the hit BALL flew far after it was hit."
illegibles2 = ["hit"]
print(find_most_frequent_word(paragraph2, illegibles2))
```

Example Output:

```
a

ball
Example 2 Explanation:
"hit" occurs 3 times, but it is an unknown word.
"ball" occurs twice (and no other word does), so it is the most frequent legible word in the
Note that words in the paragraph are not case sensitive,
that punctuation is ignored (even if adjacent to words, such as "ball,"),
and that "hit" isn't the answer even though it occurs more because it is illegible.
```

▼ 💡 **Hint: Cleaning up the String**

What string methods does Python have that might be useful for removing punctuation, spaces, standardizing capitalization, etc.? Use a search engine or check out this reference for common Python string methods to learn what helpful methods might already exist.

Problem 8: Time Portal Usage

In your time travel adventures, you have been collecting data on the usage of different time portals by various travelers. The data is represented by an array `usage_records`, where `usage_records[i] = [traveler_name, portal_number, time_used]` indicates that the traveler `traveler_name` used the portal `portal_number` at the time `time_used`.

Return the adventure's "display table". The "display table" is a table whose row entries denote how many times each portal was used at each specific time. The first column is the portal number and the remaining columns correspond to each unique time in chronological order. The first row should be a header whose first column is "Portal", followed by the times in chronological order. Note that the traveler names are not part of the table. Additionally, the rows should be sorted in numerically increasing order.

```
def display_time_portal_usage(usage_records):
    pass
```

Example Usage:

```
usage_records1 = [["David","3","10:00"],
                  ["Corina","10","10:15"],
                  ["David","3","10:30"],
                  ["Carla","5","11:00"],
                  ["Carla","5","10:00"],
                  ["Rous","3","10:00"]]
usage_records2 = [["James","12","11:00"],
                  ["Ratesh","12","11:00"],
                  ["Amadeus","12","11:00"],
                  ["Adam","1","09:00"],
                  ["Brianna","1","09:00"]]
usage_records3 = [["Laura","2","08:00"],
                  ["Jhon","2","08:15"],
                  ["Melissa","2","08:30"]]

print(display_time_portal_usage(usage_records1))
print(display_time_portal_usage(usage_records2))
print(display_time_portal_usage(usage_records3))
```

Example Output:

```
[['Portal','10:00','10:15','10:30','11:00'],['3','2','0','1','0'],['5','1','0','0','1'],
 ['10','0','1','0','0']]
[['Portal','09:00','11:00'],['1','2','0'],['12','0','3']]
[['Portal','08:00','08:15','08:30'],['2','1','1','1']]
```

▼ ✨ AI Hint: `sorted()`

Key Skill: Use AI to explain code concepts

This problem may benefit from use of the `sorted()` function. For a quick refresher on how the `sorted()` function works, check out the Unit 2 Cheatsheet.

If you'd still like to see more examples or ask follow-up questions, try using an AI tool like ChatGPT or GitHub Copilot. You can use the following prompt as a starting point:

"You're an expert computer science tutor. Please provide 2-3 examples of how the `sorted()` function is used in Python, and explain how each one works."

Close Section