Practice Interview

Objective

The partner assignment aims to provide participants with the opportunity to practice coding in an interview context. You will analyze your partner's Assignment 1. Moreover, code reviews are common practice in a software development team. This assignment should give you a taste of the code review process.

Group Size

Each group should have 2 people. You will be assigned a partner

Parts:

- Part 1: Complete 1 of 3 questions
- Part 2: Review your partner's Assignment 1 submission
- Part 3: Perform code review of your partner's assignment 1 by answering the questions below
- Part 3: Reflect on Assignment 1 and Assignment 2

Part 1:

You will be assigned one of three problems based of your first name. Enter your first name, in all lower case, execute the code below, and that will tell you your assigned problem. Include the output as part of your submission (do not clear the output). The problems are based-off problems from Leetcode.

In [61]:
```
import hashlib

def hash_to_range(input_string: str) -> int:
     hash_object = hashlib.sha256(input_string.encode())
     hash_int = int(hash_object.hexdigest(), 16)
     return (hash_int % 3) + 1
input_string = "aakash"
result = hash_to_range(input_string)
print(result)
```

```
2
```

▶ Question 1

Starter Code for Question 1

In [62]:
```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val = 0, left = None, right = None):
#         self.val = val
#         self.left = left
#         self.right = right
# def is_duplicate(root: TreeNode) -> int:
 #TODO
```

▶ Question 2

Starter Code for Question 2

In [63]:
```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val = 0, left = None, right = None):
#         self.val = val
#         self.left = left
#         self.right = right
def bt_path(root: TreeNode) -> List[List[int]]:
  # TODO
```

```
  Cell In[63], line 8
    # TODO
          ^
SyntaxError: incomplete input
```

In [ ]:
```
# Definition for a binary tree node.
class TreeNode(object):
    def __init__(self, val = 0, left = None, right = None):
        self.val = val
        self.left = left
        self.right = right
```

In [ ]:
```
from typing import List
def bt_path(root: TreeNode) -> List[List[int]]:
    result = []  # This will store ALL the paths we find
    bt_path_helper(root,[],result)
    return result

def bt_path_helper(node, current_path, result):
    # This helper function explores the tree
    # STEP 1: Check if we've hit a dead end (empty node)
    if not node:
        return  # Nothing to do, go back

    # STEP 2: Add current node to our path
    current_path.append(node.val)

    # STEP 3: Check if this is a leaf (no children)
    if not node.left and not node.right:
    # We save this path as we are at the leaf
        result.append(current_path[:])
```

```
    # STEP 4: If not a leaf, keep exploring
    else:
        bt_path_helper(node.left, current_path, result)   # going l
        bt_path_helper(node.right, current_path, result)  # going r

    # STEP 5: Backtrack - remove current node before going back
    current_path.pop()
    # This removes the last item, so we can try other paths
```

In [ ]:
```
# Create the nodes
root1 = TreeNode(10)         # Top node (value = 10)
root1.left = TreeNode(9)     # Left child of 10 is 9
root1.right = TreeNode(7)    # Right child of 10 is 7
root1.left.left = TreeNode(8)  # Left child of 9 is 8

# Tree we have:
#       10
#      /  \
#     9    7
#    /
#   8
bt_path(root1)
```

Out[ ]:
```
[[10, 9, 8], [10, 7]]
```

In [ ]:
```
# Create the nodes
root = TreeNode(1)         # Top node (value = 1)
root.left = TreeNode(2)     # Left child of 1 is 2
root.right = TreeNode(2)    # Right child of 1 is 2
root.left.left = TreeNode(3)  # Left child of 2 is 3
root.left.right = TreeNode(5)  # Right child of 2 is 5
root.right.left = TreeNode(6)  # Left child of 2 is 6
root.right.right = TreeNode(7)  # Left child of 2 is 7

# Tree we have:
#        1
#      /   \
#     2     2
#    / \   / \
#   3   5 6   7
bt_path(root)
```

Out[ ]:
```
[[1, 2, 3], [1, 2, 5], [1, 2, 6], [1, 2, 7]]
```

▶ Question 3

Starter Code for Question 3

In [ ]:
```
def missing_num(nums: List) -> int:
  # TODO
```

Part 2:

You and your partner must share each other's Assignment 1 submission.

Part 3:

Create a Jupyter Notebook, create 6 of the following headings, and complete the following for your partner's assignment 1:

- Paraphrase the problem in your own words.

In [ ]:
```
print("My partner was assigned Question 1 in Assignment 1.\n"
      "In the problem, for a given list, going left to right, we ne
      "If there is no repetitions, then function should return -1.\
      "If there are more than one number repeating in the list, we
```

```
My partner was assigned Question 1 in Assignment 1.
In the problem, for a given list, going left to right, we need to fi
nd the first element in the list that repeats. Our function should r
eturn that element.
If there is no repetitions, then function should return -1.
If there are more than one number repeating in the list, we need to
return the number which has the repetition first when going from let
to right.
```

- Create 1 new example that demonstrates you understand the problem. Trace/walkthrough 1 example that your partner made and explain it.

In [ ]:
```
test_cases_aakash = [
    [1, 2, 3, 4, 5, 6, 7], # Eg1 - no repetitions
    [6, 8, 10, 9, 7, 10], # Eg2 - one element repeats
    [2, 9, 2, 6, 7, 9]  # Eg3 - two elements repeat
]
test_cases_Ajinkya_Korade = [
    [3, 1, 4, 2, 5, 1, 6],
    [7, 8, 9, 10],
    [4, 5, 6, 4, 6]
]
```

In [ ]:
```
print('The function "first_duplicate" assesses every element in the
      'Upon finding the first instance of the repetition of an elem
      'In case no elements in the list have a duplicate present, th
      'For example in the list [3, 1, 4, 2, 5, 1, 6], going element
      '3 -> 1 -> 4 ->2 -> 5 -> 1. Here, it will see that 1 has repe
```

```
The function "first_duplicate" assesses every element in the list le
ft to right, and checks if there is any element that has a duplicate
in the list.
Upon finding the first instance of the repetition of an element in t
he list, it returns that element.
In case no elements in the list have a duplicate present, the functi
on returns -1.
For example in the list [3, 1, 4, 2, 5, 1, 6], going element by elem
ent, the function will evaluate every element from left to right
 3 -> 1 -> 4 ->2 -> 5 -> 1. Here, it will see that 1 has repeated ag
ain, then the function will stop and return the value 1
```

- Copy the solution your partner wrote.

In [ ]:
```python
# Copying partner's function to identify duplicates
from typing import List

def first_duplicate(nums: List[int]) -> int:
    hash_set = set()
    for num in nums:
        if num in hash_set:
            return num
        hash_set.add(num)
    return -1
```

In [ ]:
```python
# testing the function with Ajinkya Korade's examples
for test in test_cases_Ajinkya_Korade:
    result = first_duplicate(test)
    print(f"input_list: {test}, result: {result}")
```

```
input_list: [3, 1, 4, 2, 5, 1, 6], result: 1
input_list: [7, 8, 9, 10], result: -1
input_list: [4, 5, 6, 4, 6], result: 4
```

In [ ]:
```python
# testing the function with new examples
for test in test_cases_aakash:
    result = first_duplicate(test)
    print(f"input_list: {test}, result: {result}")
```

```
input_list: [1, 2, 3, 4, 5, 6, 7], result: -1
input_list: [6, 8, 10, 9, 7, 10], result: 10
input_list: [2, 9, 2, 6, 7, 9], result: 2
```

- Explain why their solution works in your own words.

In [ ]:
```python
print('The solution works as the function starts with defining hash
'Then the function initiates a for loop for every num in nums, then
'If the element num does not exists in the hash-set, we append the
'In the case when the for loop iterates over every element and no e
```

```
The solution works as the function starts with defining hash_set as
an empty set.
Then the function initiates a for loop for every num in nums, then w
e have a if statement checking if the element num exists in hash_se
t, we return the element num.
If the element num does not exists in the hash-set, we append the nu
m element to the hash_set.
In the case when the for loop iterates over every element and no ele
ments of the nums are returned, we return the value -1.
```

- Explain the problem's time and space complexity in your own words.

In [ ]:
```
print('The space and time complexity for this function is O(n).\n'
      'We have O(n) for both space and time complexity as each elem
      'We store every element once after it is evaluated in the has
```

```
The space and time complexity for this function is O(n).
We have O(n) for both space and time complexity as each element is i
nspected only once in the loop and
We store every element once after it is evaluated in the hash_set
```

- Critique your partner's solution, including explanation, and if there is anything that should be adjusted.

In [ ]:
```
print("The solution is optimal and well-implemented. The use of a s
      "One minor suggestion: could add a comment explaining why we
```

```
The solution is optimal and well-implemented. The use of a set is th
e right data structure choice. The explanation is thorough.
One minor suggestion: could add a comment explaining why we return i
mmediately (to ensure we get the 'first' duplicate).
```

Part 4:

Please write a 200 word reflection documenting your process from assignment 1, and your presentation and review experience with your partner at the bottom of the Jupyter Notebook under a new heading "Reflection." Again, export this Notebook as pdf.

Reflection

In [ ]:
```
print("""
Reflections

Assignment 1 taught me the importance of choosing the appropriate a
For my problem on finding the valid bracket sequence, employing sta
The Stacks data structure provides access to the most recent item.
Specifying the bracket_map was very important as in a valid bracket
when we have a "closing bracket", it must match the most recently "

For Assignment 2 Part 1, I was assigned the path-to-leaves problem.
```

```
me to understand tree traversal and recursion deeply. After tracing
through examples on paper and referring to live code notebook, I us

My partner Ajinkya's work is very detailed and meticulous. I apprec
Their solution for finding duplicates was nearly identical to what
The hash_set approach was elegant and optimal.

This code review process gives me a taste for real software develop
explaining code clearly is just as important as writing it correctl
""")
```

```
Reflections

Assignment 1 taught me the importance of choosing the appropriate ab
stract data structure.
For my problem on finding the valid bracket sequence, employing stac
k data structure - with a Last-In-First-Out (LIFO) principle was app
ropriate.
The Stacks data structure provides access to the most recent item.
Specifying the bracket_map was very important as in a valid bracket
sequence,
when we have a "closing bracket", it must match the most recently "o
pening bracket".

For Assignment 2 Part 1, I was assigned the path-to-leaves problem.
This challenged
me to understand tree traversal and recursion deeply. After tracing
through examples on paper and referring to live code notebook, I use
d the helper function to implement my solution.

My partner Ajinkya's work is very detailed and meticulous. I appreci
ate the structure of their explaination provided.
Their solution for finding duplicates was nearly identical to what I
would have written.
The hash_set approach was elegant and optimal.

This code review process gives me a taste for real software developm
ent process. I learned that
explaining code clearly is just as important as writing it correctl
y. It is a valuable skill I'll carry forward.
```

Evaluation Criteria

We are looking for the similar points as Assignment 1

- Problem is accurately stated
- New example is correct and easily understandable
- Correctness, time, and space complexity of the coding solution
- Clarity in explaining why the solution works, its time and space complexity
- Quality of critique of your partner's assignment, if necessary

Submission Information

🚨 **Please review our Assignment Submission Guide** 🚨 for detailed instructions on how to format, branch, and submit your work. Following these guidelines is crucial for your submissions to be evaluated correctly.

Submission Parameters:

- Submission Due Date: `HH:MM AM/PM - DD/MM/YYYY`
- The branch name for your repo should be: `assignment-2`
- What to submit for this assignment:
  - This Jupyter Notebook (assignment_2.ipynb) should be populated and should be the only change in your pull request.
- What the pull request link should look like for this assignment: `https://github.com/<your_github_username>/algorithms_and_da`
  - Open a private window in your browser. Copy and paste the link to your pull request into the address bar. Make sure you can see your pull request properly. This helps the technical facilitator and learning support staff review your submission easily.

Checklist:

- ☐ Created a branch with the correct naming convention.
- ☐ Ensured that the repository is public.
- ☐ Reviewed the PR description guidelines and adhered to them.
- ☐ Verify that the link is accessible in a private browser window.

If you encounter any difficulties or have questions, please don't hesitate to reach out to our team via our Slack at `#cohort-6-help`. Our Technical Facilitators and Learning Support staff are here to help you navigate any challenges.

In [ ]: