Author: Madhurima Rawat

# Pair Programming Activity

**Pair up students to work on a programming task collaboratively, taking turns as the driver and navigator, and switch roles periodically to promote teamwork and knowledge sharing.**

## What is Pair Programming?

Pair programming is a collaborative programming technique where two developers work together on a single task. They alternate between two roles:

1. **Driver:** Actively writes the code, focusing on implementation.
2. **Navigator:** Observes the code, reviews it in real time, and provides feedback or suggestions.

Switching roles periodically helps both participants engage actively, share knowledge, and improve their programming skills.

Benefits of pair programming include:

- **Improved Code Quality:** Real-time feedback minimizes bugs and improves design.
- **Knowledge Sharing:** Enhances learning by exposing participants to different approaches and ideas.
- **Teamwork Development:** Strengthens collaboration and communication skills.

In this activity, pairs of students will work together to implement a function to determine whether a given string is a palindrome.

## Scenario and Participants

- **Team 1:** Alice (Driver) and Bob (Navigator)
- **Team 2:** Bob (Driver) and Charlie (Navigator)
- **Team 3:** Charlie (Driver) and Alice (Navigator)

Each pair will write, review, and improve a function for determining if a string is a palindrome, considering edge cases such as spaces, punctuation, and mixed case.

## Code and Reviews

## Team 1: Alice and Bob

**Initial Code (Alice as Driver):**

```python
def is_palindrome(s):
    s = s.lower()
    return s == s[::-1]
```

**Bob's Suggestions as Navigator:**

- Suggestion: Remove non-alphanumeric characters before checking for palindrome status.
- Suggestion: Add a docstring for clarity.
- Suggestion: Add test cases for edge cases like "A man, a plan, a canal, Panama."

**Updated Code:**

```python
def is_palindrome(s: str) -> bool:
    """
    Check if a given string is a palindrome.

    Args:
        s (str): The string to check.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    s = ''.join(c.lower() for c in s if c.isalnum())
    return s == s[::-1]
```

## Team 2: Bob and Charlie

**Initial Code (Bob as Driver):**

```python
def check_palindrome(string):
    return string.replace(" ", "").lower() == string.replace(" ", "").lower()[::-1]
```

**Charlie's Suggestions as Navigator:**

- Suggestion: Handle punctuation and special characters.
- Suggestion: Replace the function name with `is_palindrome` for consistency.
- Suggestion: Add type hints and a docstring.

**Updated Code:**

```python
def is_palindrome(s: str) -> bool:
    """
    Determine whether the given string is a palindrome.

    Args:
        s (str): The string to check.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    cleaned = ''.join(c.lower() for c in s if c.isalnum())
    return cleaned == cleaned[::-1]
```

## Team 3: Charlie and Alice

### Initial Code (Charlie as Driver):

```python
def palindrome(s):
    s = ''.join(s.split()).lower()
    return s == s[::-1]
```

### Alice's Suggestions as Navigator:

- Suggestion: Rename the function to `is_palindrome` for consistency.
- Suggestion: Use a list comprehension to remove non-alphanumeric characters.
- Suggestion: Add a docstring and type hints.

### Updated Code:

```python
def is_palindrome(s: str) -> bool:
    """
    Verify if the input string is a palindrome, ignoring spaces and case.

    Args:
        s (str): Input string to check.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    s = ''.join(c for c in s.lower() if c.isalnum())
    return s == s[::-1]
```

## Conclusion

This **Pair Programming Activity** demonstrated the power of collaboration in software development. Key takeaways include:

1. **Enhanced Code Quality:** Real-time feedback led to cleaner and more robust solutions.
2. **Diverse Perspectives:** Each participant contributed unique insights to improve the code.
3. **Teamwork and Communication:** Alternating roles ensured active participation and fostered mutual understanding.