Author: Madhurima Rawat

Code Review Simulation

Conduct a simulated code review session where students provide feedback on each other's code, focusing on readability, maintainability, and adherence to coding standards.

What is Code Review?

Code review is a collaborative process where peers examine each other's code to ensure it adheres to **best practices** for coding. The focus is on improving the **readability**, **maintainability**, and **functionality** of the code while fostering a culture of learning and constructive feedback.

Key benefits of code review include:

- **Identifying Bugs:** Catching errors early in the development process.
- **Improving Quality:** Ensuring code follows standards and is efficient.
- **Knowledge Sharing:** Encouraging collaboration and learning among team members.

In this simulation, three students—Alice, Bob, and Charlie—have written programs to calculate the factorial of a number. Their peers will review the code and suggest improvements, which will then be incorporated into updated versions.

Scenario and Participants

- **Alice's Code:** Uses recursion to calculate the factorial of a number.
- **Bob's Code:** Uses an iterative approach with a loop.
- **Charlie's Code:** A concise one-liner recursive function.

Each student's code will be reviewed by the other two participants, focusing on:

1. **Readability:** Is the code easy to understand?
2. **Maintainability:** Is it structured for future updates or changes?
3. **Adherence to Standards:** Does it follow Python best practices (naming, type hints, etc.)?

Code and Reviews

Alice's Code

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Bob's Review:

- The code is concise and demonstrates recursion well.
- Suggestion: Add type hints for clarity about the input and output types.
- Suggestion: Add error handling to manage cases where the input is negative.

Charlie's Review:

- The recursive logic is solid.
- Suggestion: Include a docstring to explain the function.
- Suggestion: Handle non-integer inputs gracefully.

Updated Code (Alice):

```
def factorial(n: int) -> int:
    """
    Calculate the factorial of a number.

    Args:
        n (int): The number to calculate factorial for. Must be non-negative.

    Returns:
        int: The factorial of the number.
    """
    if not isinstance(n, int) or n < 0:
        raise ValueError("Input must be a non-negative integer.")
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Bob's Code

```
def fact(n):
    f = 1
    for i in range(1, n + 1):
        f *= i
    return f
```

Alice's Review:

- Suggestion: Rename the function to `factorial` for a more descriptive name.
- Suggestion: Add comments to explain the purpose of the loop and overall functionality.

Charlie's Review:

- The iterative approach is clear and avoids recursion limits.
- Suggestion: Add input validation to ensure non-negative integers are passed.
- Suggestion: Include a docstring and type hints for better documentation.

Updated Code (Bob):

```python
def factorial(n: int) -> int:
    """
    Compute the factorial of a non-negative integer.

    Args:
        n (int): The number to compute factorial for.

    Returns:
        int: The factorial of the number.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    f = 1
    for i in range(1, n + 1):
        f *= i
    return f
```

Charlie's Code

```python
def fact(n):
    return 1 if n == 0 else n * fact(n - 1)
```

Alice's Review:

- Suggestion: Break the one-liner into a multi-line implementation for readability.
- Add a docstring to clarify the function's purpose and behavior.

Bob's Review:

- The recursive approach is correct and efficient.
- Suggestion: Rename the function to `factorial` for consistency.
- Suggestion: Add input validation to prevent invalid inputs from causing errors.

Updated Code (Charlie):

```python
def factorial(n: int) -> int:
    """
    Calculate the factorial of a number.

    Args:
        n (int): A non-negative integer.

    Returns:
        int: The factorial of the number.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    return 1 if n == 0 else n * factorial(n - 1)
```

Conclusion

This exercise showcased the importance of **collaboration** and **feedback** in improving code quality. The updated versions demonstrate:

1. **Improved Readability:** Descriptive names, docstrings, and comments make the code easier to understand.
2. **Enhanced Maintainability:** Input validation and consistent structure prepare the code for future enhancements.
3. **Standards Adherence:** Adding type hints and following Python conventions ensures clarity and reliability.

By fostering a culture of **constructive criticism**, code reviews help individuals and teams produce more robust and professional software.