Below are five Clean Code principles applied to the MaswaliYote Quiz Management System, along with explanations of how they improve the code:

1. Meaningful Variable Names:

Original Code:

```
def add_quiz(q, q_list):
    q_list.append(q)
```

Improved Code:

```
def add_quiz(quiz, quiz_list):
    quiz_list.append(quiz)
```

Explanation:
The improved code uses meaningful variable names ("quiz" instead of "q" and "quiz_list" instead of "q_list"). This makes the code self-explanatory and enhances readability. Maintainers can quickly understand the purpose of the variables without having to decipher cryptic names.

2. Descriptive Function and Method Names:

Original Code:

```
def f1():
    # function logic here
```

Improved Code:

```
def create_quiz():
    # function logic here
```

Explanation:
Descriptive function and method names provide clarity about the purpose of the code. In the improved code, "create_quiz" clearly communicates the intent of the function, making it easier for others to understand the functionality without delving into the implementation details.

3. Avoiding Magic Numbers:

Original Code:

```
if student_score > 70:
    # logic here
```

Improved Code:

```
PASSING_SCORE_THRESHOLD = 70

if student_score > PASSING_SCORE_THRESHOLD:
    # logic here
```

Explanation:
The improved code replaces the magic number "70" with a named constant ("PASSING_SCORE_THRESHOLD"). This enhances code readability by providing context for the numeric value. If the passing score changes in the future, it only needs to be updated in one place.

4. Single Responsibility Principle (SRP):

Original Code:

```
def process_quiz_data(data):
    # logic to process quiz data
    # logic to store results
```

Improved Code:

```
def process_quiz_data(data):
    # logic to process quiz data

def store_quiz_results(results):
    # logic to store results
```

Explanation:
The improved code adheres to the Single Responsibility Principle by separating the concerns of processing quiz data and storing results into two distinct functions. This results in more modular and maintainable code.

5. Consistent Code Formatting:

Original Code:

```
def add_student_result(result, student_results):
result['timestamp'] = datetime.now()
student_results.append(result)
```

Improved Code:

```
def add_student_result(result, student_results):
    result['timestamp'] = datetime.now()
    student_results.append(result)
```

Explanation:
Consistent code formatting, such as proper indentation and use of whitespace, enhances readability. The improved code follows a consistent formatting style, making it easier to follow the logical structure of the code.

Clean Code Cheatsheet:

Here's a quick cheatsheet summarizing Clean Code principles:

1. Meaningful Variable Names:
   - Use names that reveal intent.
   - Avoid single-letter names unless they are iterators.

2. Descriptive Function and Method Names:
   - Choose names that clearly convey the purpose of the function or method.
   - Avoid generic names like `process` or `calculate`.

3. Avoiding Magic Numbers:
   - Replace magic numbers with named constants.
   - Provide context for numeric values.

4. Single Responsibility Principle (SRP):
   - A function/method should have a single, clear responsibility.
   - Separate concerns into distinct functions/methods.

5. Consistent Code Formatting:
   - Use consistent indentation.
   - Follow a consistent style for braces, line breaks, and spacing.