import ast

class CodeSmellDetector(ast.NodeVisitor):

    def \_\_init\_\_(self):

        self.smells = []  # List to store identified code smells

    def visit\_FunctionDef(self, node):

        if len(node.name) < 3:  # Check for short method names

            self.smells.append(f"Rename Method: Method '{node.name}' is too short and unclear.")

        if any(char.isdigit() for char in node.name):  # Check for numeric characters in method names

            self.smells.append(f"Rename Method: Method '{node.name}' contains numeric characters, making it unclear.")

        if "\_" in node.name:  # Check for underscores indicating possible unclear naming

            self.smells.append(f"Rename Method: Method '{node.name}' contains underscores, which can reduce clarity.")

        # Continue to traverse the AST

        self.generic\_visit(node)

    def report\_smells(self):

        if not self.smells:

            return "No significant code smells detected."

        return "\n".join(self.smells)

def analyze\_code\_for\_refactoring(source\_code):

    # Parse the source code into an AST

    tree = ast.parse(source\_code)

    # Create an instance of the CodeSmellDetector

    detector = CodeSmellDetector()

    # Visit the AST nodes to check for code smells

    detector.visit(tree)

    # Return the detected code smells

    return detector.report\_smells()

# Example Usage

source\_code = """

class Employee:

    def \_\_init\_\_(self, name, salary):

        self.name = name

        self.salary = salary

    def doStuff():  # Ambiguous name

        pass

    def computeBonus(self):  # Clearer name

        return self.salary \* 0.1

    def func123(self):  # Contains digits

        return self.salary

    def \_calculate(self):  # Underscore indicating possible internal use

        return self.salary \* 0.1

"""

# Analyze the source code for refactoring needs

refactoring\_report = analyze\_code\_for\_refactoring(source\_code)

print(refactoring\_report)