Listing 1: Example Python Code

```python
#!/usr/bin/env python3
#!/usr/bin/env python3


def multiply_matrices(a: list, b: list) -> list:
    """
    Multiply two 2x2 matrices.

    Args:
        a (list): The first matrix.
        b (list): The second matrix.

    Returns:
        list: The product of the two matrices.

    Raises:
        TypeError: If either matrix is not a list of lists.
        ValueError: If either matrix is not 2x2 or if the
            elements are not numbers.
    """
    if not isinstance(a, list) or not isinstance(b, list):
        raise TypeError("Both matrices must be lists of lists")
    if len(a) != 2 or len(b) != 2 or len(a[0]) != 2 or
        len(b[0]) != 2:
        raise ValueError("Both matrices must be 2x2")
    if not all(isinstance(row, list) for row in a) or not all(
        isinstance(row, list) for row in b
    ):
        raise TypeError("Both matrices must be lists of lists")
    if not all(
        isinstance(element, (int, float)) for row in a for
            element in row
    ) or not all(isinstance(element, (int, float)) for row in
        b for element in row):
        raise ValueError("All elements of the matrices must be
            numbers")

    result = [
        [sum(a_val * b_val for a_val, b_val in zip(a_row,
            b_col)) for b_col in zip(*b)]
        for a_row in a
    ]
    if not isinstance(result, list) or not all(isinstance(row,
        list) for row in result):
        raise TypeError("The result must be a list of lists")
    if not all(isinstance(element, (int, float)) for row in
        result for element in row):
        raise ValueError("All elements of the result must be
```

```
            numbers")
    return result


def matrix_power(matrix: list, n: int) -> list:
    """
    Calculate the nth power of a matrix using a
        divide-and-conquer approach.

    Args:
        matrix (list): The matrix to exponentiate.
        n (int): The exponent.

    Returns:
        list: The nth power of the matrix.

    Raises:
        TypeError: If the matrix is not a list of lists or if
            the exponent is not an integer.
        ValueError: If the exponent is negative or if the
            matrix is not 2x2 or if the elements are not
            numbers.
    """
    if not isinstance(matrix, list):
        raise TypeError("The matrix must be a list of lists")
    if not isinstance(n, int):
        raise TypeError("The exponent must be an integer")
    if n < 0:
        raise ValueError("Exponent must be a non-negative
            integer")
    if len(matrix) != 2 or len(matrix[0]) != 2:
        raise ValueError("The matrix must be 2x2")
    if not all(isinstance(row, list) for row in matrix):
        raise TypeError("The matrix must be a list of lists")
    if not all(isinstance(element, (int, float)) for row in
        matrix for element in row):
        raise ValueError("All elements of the matrix must be
            numbers")

    if n == 0:
        result = [[1, 0], [0, 1]]
    elif n == 1:
        result = matrix
    elif n % 2 == 0:
        half_pow = matrix_power(matrix, n // 2)
        result = multiply_matrices(half_pow, half_pow)
    else:
        half_pow = matrix_power(matrix, n // 2)
        result = multiply_matrices(multiply_matrices(half_pow,
            half_pow), matrix)
```

```
    if not isinstance(result, list) or not all(isinstance(row, list) for row in result):
        raise TypeError("The result must be a list of lists")
    if not all(isinstance(element, (int, float)) for row in result for element in row):
        raise ValueError("All elements of the result must be numbers")
    return result


def fibonacci(n: int) -> int:
    """
    Calculate the nth Fibonacci number using matrix exponentiation.

    Args:
        n (int): The index of the Fibonacci number to calculate.

    Returns:
        int: The nth Fibonacci number.

    Raises:
        TypeError: If the index is not an integer.
        ValueError: If the index is negative.
    """
    if not isinstance(n, int):
        raise TypeError("The index must be an integer")
    if n < 0:
        raise ValueError("Index must be a non-negative integer")
    if n == 0:
        return 0
    fib_matrix = [[1, 1], [1, 0]]
    result_matrix = matrix_power(fib_matrix, n - 1)
    result = result_matrix[0][0]
    if not isinstance(result, int):
        raise TypeError("The result must")
    return result


if __name__ == "__main__":
    n = 10
    print(f"fibonacci({n})={fibonacci(n)}")
```

hello 1